Sistemas Informáticos
Curso 2.004-05

Módulo de Inteligencia Artificial para el proyecto GICE4S

Manuel Fuertes Marín
Patricia Gallo Martínez
Natalia Ortiz de Zárate Ansótegui

Dirigido por:
Prof. Belén Díaz Agudo
Dpto. Sistemas Informáticos y Programación

Facultad de Informática
Universidad Complutense de Madrid

AUTORIZACIÓN

Los alumnos abajo firmantes autorizan a la Universidad Complutense de Madrid a difundir y utilizar con fines académicos, no comerciales y mencionando expresamente a sus autores, tanto la propia memoria, como el código, la documentación y el prototipo creado.

Manuel	Patricia	Natalia
Fuertes Marín	Gallo Martínez	Ortiz de Zárate Ansótegui

RESUMEN EN CASTELLANO

Este proyecto es un módulo de inteligencia artificial para el sistema GICE4S, que tiene en esta primera fase el objetivo de desarrollar un prototipo para la gestión del programa de intercambio Sócrates/Erasmus.

El prototipo dará soporte a la gestión de los distintos procesos del programa Erasmus a través de una serie de funcionalidades que atiendan los objetivos de los usuarios principales, como alumnos y coordinador.

Nuestro proyecto ofrece tres asistentes: para la elección de universidad, para la elección de asignaturas y para ayudar al coordinador a decidir si una asignatura es convalidable.

RESUMEN EN INGLÉS

This project is an artificial intelligence module for the GICE4S system that, in this first phase, has the aim of developing a prototype for the Sócrates/Erasmus exchange program management.

This prototype will support the different processes of Erasmus program through several funcionalities that support the requirements of the main users as students and coordinator.

It offers three assistants: for the university choice, for the subjects choice and one to help the coordinator to decide if a subject can be confirmed by another one.

PALABRAS CLAVE

1. Inteligencia artificial
2. CBR
3. Ontología
4. Similitud
5. GICE4S
6. Sócrates/Erasmus
7. J2EE

ÍNDICE

Página

PRÓLOGO

Entendemos la Informática como una herramienta que debe estar de acuerdo con las necesidades del usuario y partiendo de esta premisa nos sedujo la idea de desarrollar un servicio orientado a facilitar la gestión a los alumnos universitarios que quisieran optar a una beca Erasmus, ya que hasta el momento carecían de la ayuda necesaria para la elección de la universidad de destino, así como de las asignaturas que más se adaptasen a sus necesidades académicas.

Nuestras motivaciones para haber realizado este proyecto es nuestro interés por la inteligencia artificial, así como el desarrollo de aplicaciones que utilicen nuevas tecnologías relacionadas con Internet.

Otro incentivo importante para nosotros era el hecho de que el módulo de inteligencia que ibamos a realizar se pudiera a integrar en un proyecto más grande cuyo desarrollo se lleva a cabo por la Facultad de Informática, en colaboración con Bull España, al amparo del artículo 83 de la LOU. Este proyecto recibe el nombre de GICE4S (Gestión Informátizada de Centros Educativos en el Espacio Europeo de Educación Superior) y simplemente gestiona el alta de usuarios y otros aspectos administrativos sobre los trámites de la beca Erasmus. Al integrar nuestro módulo en este proyecto se le dota de inteligencia para aportar asistentes que ayudarán tanto a alumnos como al coordinador Erasmus en distintas etapas del uso de la aplicación.

CAPÍTULO I:

INTRODUCCIÓN

INTRODUCCIÓN

El proyecto GICE4S (Gestión Informatizada de Centros Educativos en el Espacio Europeo de Educación Superior) tiene en esta primera fase el objetivo de desarrollar un prototipo para la gestión del programa de intercambio Sócrates/Erasmus en un centro superior, Facultad o Escuela Universitaria.

El objetivo del programa Sócrates/Erasmus es promover la movilidad de estudiantes y profesores entre universidades de la Unión Europea.

Este proyecto se plantea en el marco del desarrollo de un prototipo para la gestión del programa de intercambio Sócrates/Erasmus en una Facultad universitaria.

Un estudiante Erasmus es seleccionado en su centro de origen, acuerda un plan de estudios y se traslada a la universidad de destino donde cursa el citado plan por el que luego se le convalidan las asignaturas que se hubiese acordado. El alumno Erasmus no se matricula en la Universidad de destino sino en la de origen.

El prototipo dará soporte a la gestión de los distintos procesos del programa Erasmus, ofreciendo una serie de funcionalidades que atiendan los objetivos de los usuarios principales, alumnos y coordinador.

La aplicación puede verse en funcionamiento en la dirección Web http://gice4s.fdi.ucm.es/MainPage.do, y presenta el siguiente aspecto:

Figura 1: Captura de la página principal del sistema GICE4S

El objetivo marcado en el proyecto de la asignatura de Sistemas Informáticos es el desarrollo de un módulo de Inteligencia Artificial, que complete las necesidades de razonamiento para el correcto funcionamiento de esta versión del proyecto GICE4S.

Las líneas principales de inteligencia desarrolladas, a grandes rasgos, son las siguientes:

1. El sistema debe ofrecer al alumno una ayuda interactiva para la elección de la universidad de destino, mostrándole un ranking de las mismas, así como las asignaturas que más se adapten a sus necesidades.

   El proceso se realiza atendiendo a una serie de preferencias seleccionadas por el usuario a través del sistema, y una serie de restricciones sobre las convalidaciones de las asignaturas que desee cursar en el intercambio Erasmus.

2. Dada una asignatura debe sugerir con que otras podría ser convalidada, en función del número de créditos y los temarios de las mismas.

   Esta funcionalidad tiene su utilidad tanto para el alumno como para el coordinador Erasmus.

   Al primero se le mostraría al igual que antes un ranking que sirva de orientación. Para el caso del coordinador le ayudaría a decidir si una convalidación puede o no efectuarse.

El módulo de inteligencia del sistema tiene como base la utilización de una función de similitud entre asignaturas. Dicha función realiza los cálculos atendiendo al contenido y al número de créditos que tienen las distintas asignaturas, siendo realmente importante para todos los casos de uso que han sido desarrollados dentro del proyecto de Sistemas Informáticos, para formar parte del sistema GICE4S.

En los siguientes capítulos se comentarán las distintas fases en las que se ha dividido el desarrollo del proyecto. El capítulo II narra la especificación de requisitos y los objetivos iniciales del proyecto, así como los alcanzados finalmente. En el capítulo III se tratará la fase de investigación, en la que se cuenta las tecnologías consideradas y las que finalmente se han utilizado. El capítulo IV aborda la fase de adquisición del conocimiento, donde se explican las distintas fuentes de conocimiento. En el capítulo V aparecen los detalles de la fase de implementación de las distintas aplicaciones que forman parte del proyecto. Y, por último, el capítulo VI contine las concluiones alcanzadas tras la finalización del proyecto.

CAPÍTULO II:

ESPECIFICACIÓN DE REQUISITOS Y OBJETIVOS INICIALES

A. INTRODUCCIÓN

Este capítulo está dedicado a la especificación de requisitos, así como a los objetivos planteados inicialmente y los objetivos cumplidos finalmente.

B. OBJETIVOS INICIALES

El proyecto de Sistemas Informáticos surge de la necesidad del desarrollo de un módulo que implemente la parte de Inteligencia Artificial del sistema.

Concretamente, el proyecto se encarga de la Inteligencia Artificial del módulo de gestión de reconocimiento académico, utilizando técnicas de razonamiento basado en casos y ontologías.

El soporte inteligente que el sistema daría al proceso de reconocimiento académico consistiría inicialmente en lo siguiente:

- Gestionar un histórico de las convalidaciones realizadas en cursos anteriores que pueda servir como base a las nuevas propuestas.

- Permitir la realización de búsquedas de asignaturas por temas. Esta opción resulta muy interesante no sólo para la definición de un acuerdo de reconocimiento de créditos sino también para ayudar al alumno a elegir Universidad. Hasta el momento, el alumno elige Universidad basándose principalmente en el idioma y lugar de destino, pero sin tener en cuenta sus necesidades académicas.

- Gestión y representación del conocimiento del dominio.

- Calcular la similitud de asignaturas de distintos centros. Para ello, el cómputo de la similitud se basará en el conocimiento del dominio y deberá tener en cuenta la heterogeneidad de los contenidos de las asignaturas, la diferencia de horas o créditos, programas de asignaturas en distintos idiomas, carácter de la asignatura (troncal, optativo, libre elección),…

C. OBJETIVOS CUMPLIDOS

El soporte inteligente desarrollado en el proyecto de Sistemas Informáticos gira en torno a la implementación de una función de similitud que ha permitido el desarrollo de distintas funcionalidades de la aplicación.

Los distintos módulos realizados durante el desarrollo del proyecto son los siguientes:

- Ayuda al alumno para elegir Universidad teniendo en cuenta sus preferencias y necesidades académicas.

- Ayuda al alumno para la convalidación de asignaturas en función de aquellas que tiene pendientes.

- Desarrollo de una ayuda dirigida al coordinador Erasmus para llevar a cabo la comprobación de la convalidación de asignaturas realizadas por el alumno.

- Desarrollo de una aplicación dedicada a la recuperación automática de información (RI). Esta aplicación resulta muy interesante para la asignación de los descriptores que cubre una asignatura de forma automática y que será necesaria a la hora de calcular la similitud entre las asignaturas.

- Desarrollo de una aplicación dedicada a mejorar la eficiencia del proyecto a la hora de cargar la ontología de la Informática, necesaria para calcular la similitud de las asignaturas.

Alguno de los objetivos iniciales no se ha cubierto, como es el caso de la gestión de un histórico de convalidaciones, debido a que se ha dado preferencia a otros objetivos no planteados en principio. Por el contrario se han implementado objetivos que surgieron durante el desarrollo de proyecto, como la aplicación dedicada a la recuperación automática de información (RI) y la que genera el archivo “ontologia.dat” con la información necesaria para calcular la similitud de asignaturas, consiguiendo con ello una mejora de la eficiencia del proyecto.

D. ESPECIFICACIÓN DE REQUISITOS

El proyecto GICE4S tiene el objetivo de desarrollar un prototipo para la gestión del programa de intercambio Erasmus en un centro superior, Facultad o Escuela Universitaria.

El objetivo del programa Erasmus es promover la movilidad de estudiantes y profesores entre universidades de la Unión Europea. Un estudiante Erasmus es seleccionado en su centro de origen, acuerda un plan de estudios y se traslada a la universidad de destino donde cursa el citado plan por el que luego se le convalidan las asignaturas que se hubiese acordado. El alumno Erasmus no se matricula en la Universidad de destino sino en la de origen.

1. Actores principales y sus objetivos

- Coordinador Erasmus Institucional

  Es el responsable del programa Erasmus en la Facultad. Su objetivo general es el buen funcionamiento del programa y ha de llevar a cabo los siguientes procesos ordenados cronológicamente:

  - Firma de acuerdos de intercambio. El plazo es hasta octubre.
  - Convocatoria Erasmus. El plazo es hasta Diciembre.
  - Exámenes de idioma. Del 15 de febrero al 15 de marzo.
  - Preselección. Tiene lugar en la segunda quincena de marzo.
  - Asignación de plazas. Tiene lugar en el mes de Abril
  - Gestión de documentos internos. En los meses de Mayo y junio
  - Gestión de documentos externos. Tiene lugar en el mes Junio.
  - Aceptación de estudiantes extranjeros. Durante los mese de Junio y julio.
  - Gestión de las calificaciones de los estudiantes in. Durante los mese de Junio y Septiembre.
  - Gestión de las convalidaciones. Durante los mese de Julio y Septiembre.
  - Recepción de los estudiantes in. Durante el mes de Septiembre.

- Coordinador Erasmus Departamental

  Es un colaborador del Coordinador Erasmus Institucional. Su objetivo es comprobar la validez del acuerdo de reconocimiento académico y asesorar al Estudiante Erasmus Out. En un mismo centro hay varios coordinadores departamentales, cada uno de los cuales es responsable del intercambio con una o más universidades.

  El Coordinador Erasmus Departamental también asesora al Estudiante Erasmus Out en el proceso de Gestión del acuerdo de reconocimiento académico. Se empieza con la selección del estudiante Erasmus en Mayo, y puede sufrir modificaciones durante toda la estancia del estudiante, típicamente en Septiembre y Febrero.

- Estudiante Interesado Out

  Estudiante de la Universidad de origen interesado en ser estudiante Erasmus para cursar sus estudios el próximo año en una Universidad extranjera. Este estudiante presenta la solicitud Erasmus en la Universidad de origen.

- Estudiante Erasmus Out

  Estudiante que ha conseguido una beca para ir a estudiar a una Universidad extranjera.

  Un Estudiante Erasmus Out antes ha pasado por el estado de Estudiante Interesado Out. Debe realizar los siguientes procesos:

  - Cumplimentar el formulario oficial de inscripción Erasmus.
  - Preparar su acuerdo de estudios antes de marcharse y obtener la autorización de los coordinadores departamental e institucional.
  - Cumplimentar los documentos que solicite la universidad de destino.
  - Recoger en la Oficina Erasmus las certificaciones que le acreditan como estudiante Erasmus, el certificado de llegada y el de salida, necesarios para percibir la beca.
  - Pactar con el coordinador departamental cualquier modificación al acuerdo de estudios.

2. Procesos

- Convocatoria Erasmus

  A finales de Diciembre o principios de Enero, se hace pública la convocatoria Erasmus para el curso próximo y se convoca a los estudiantes a una reunión informativa. En la convocatoria se incluyen distintos apartados, algunos fijos y otros variables de una convocatoria a otra:

  - Información general sobre el programa. Esta información no debe variar mucho de una convocatoria a otra.
  - Guía general de trámites que habrá de realizar un estudiante Erasmus, una vez seleccionado como tal.
  - Plazas convocadas este curso. Esta lista se obtiene de la lista de contratos firmados para este curso.

- Procedimiento de solicitud de una plaza erasmus:
    - Presentación de un impreso de solicitud.
    - Prueba de idioma.
    - Elección de plaza según la puntuación obtenida.

- Exámenes de idioma

  Cada estudiante tiene derecho a realizar dos pruebas de idioma que le sumarán puntos para las universidades en las cuales se impartan las clases en esos idiomas. A partir de los impresos de solicitud se ha de generar una lista con los estudiantes que se presentan a cada examen y remitirla a la oficina Erasmus. Transcurrido un cierto tiempo la oficina Erasmus devuelve las fechas de los exámenes de idioma que es necesario comunicar a los alumnos.

  Una vez realizados los exámenes la oficina Erasmus envía las calificaciones de los estudiantes que es necesario incorporar a sus expedientes.

- Preselección

  Se realiza a partir de las solicitudes, una vez que se tiene la información necesaria para aplicar el baremo. Este baremo suele tener en cuenta la nota media de la carrera, el número de créditos cursados hasta el momento, la calificación en la prueba de idioma y otros méritos que los alumnos hayan hecho constar en su solicitud como títulos de idiomas, cursos realizados o experiencia laboral.

- Gestión de las convalidaciones

  Cuando vuelve el Estudiante Erasmus Out, en Junio o en Septiembre, llegará con las certificaciones de las asignaturas que haya cursado en la universidad extranjera. En ese momento, a partir del acuerdo de reconocimiento académico al que se haya llegado se genera el acta de equivalencia donde se recogen formalmente las asignaturas que se convalidan por la estancia Erasmus.

- Gestión del acuerdo de reconocimiento académico

  El acuerdo de reconocimiento académico especifica qué asignaturas cursará un estudiante Erasmus durante su estancia y qué asignaturas le serán convalidadas cuando regrese. Por una parte, obliga al estudiante a planificar su estancia.

  Actualmente el proceso de definición del acuerdo se realiza artesanalmente. El estudiante analiza el plan de estudios de la universidad de destino y, de acuerdo con las asignaturas que tiene

pendientes, selecciona cuales de ellas quiere cursar y por cuáles le gustaría convalidar. A continuación, presenta esta propuesta al coordinador departamental que es el responsable de darle el visto bueno. Esta propuesta puede sufrir varias iteraciones hasta llegar a un acuerdo por parte de ambos.

Existen algunas directrices generales a la hora de confeccionar el acuerdo:

- La equivalencia entre asignaturas, tanto por contenidos como por número de créditos, ha de ser bastante estricta para las asignaturas troncales y obligatorias, relajada para las asignaturas optativas y muy relajada para asignaturas de libre elección.

- Aunque existe un sistema europeo de intercambio de créditos, el sistema ECTS (European Credit Transfer System), no todas las universidades lo han adoptado, y en muchos casos es necesario realizar algún tipo de regla de tres que se basa en el número de créditos de un cursos académico: 60 ECTS.

CAPÍTULO III:

FASE DE INVESTIGACIÓN

A. INTRODUCCIÓN

Este capítulo está dedicado a la fase de investigación realizada para conocer las tecnologías y aplicaciones utilizadas en el proyecto. Se contará sucintamente en que consisten pero nos centraremos en el porqué se han elegido éstas y no otras igualmente posibles considerando las ventajas e inconvenientes que conllevan.

Esta fase de investigación se desarrolló principalmente en la etapa inicial del proyecto pero se ha extendido a lo largo de todo él a medida que han ido surgiendo las distintas alternativas.

B. JAVA 2 PLATFORM, ENTERPRISE EDITION (J2EE)

Al desarrollar un módulo de inteligencia artificial para un proyecto mayor que ha utilizado J2EE, como es el caso de GICE4S, hemos tenido que hacerlo también utilizando este estándar. Por lo tanto el uso de esta tecnología ha sido una decisión que no hemos tenido que tomar puesto que no había otra opción. En cualquier caso, es una tecnología muy apropiada para el desarrollo de la aplicación porque se ajusta adecuadamente a las características de la misma.

Para la utilización de este estándar y el desarrollo del proyecto se ha utilizado el servidor de aplicaciones JOnAS y el entorno de desarrollo Eclipse, ambos de Bull.

En la actualidad, la necesidad de desarrollar aplicaciones distribuidas que trabajen en forma transaccional, conservando niveles de rapidez, seguridad y escalabilidad, han ido aumentando. La estructura tradicional de dos capas (cliente-servidor) ha ido evolucionando en estructuras más complejas, formadas por más capas, en las cuales se busca una división clara de los roles que forman parte de las componentes de las aplicaciones, contribuyendo a una mejor reutilización, crecimiento y mantenimiento de los sistemas existentes.

J2EE define el estándar para el desarrollo de aplicaciones multihilo. Esta plataforma simplifica las aplicaciones de la empresa basándose en componentes estandartizados, modulares, proporcionando un juego completo de servicios a éstos, y manejando muchos detalles del comportamiento de la aplicación automáticamente, evitando que el desarrollador tenga que hacerlo con un código complejo.

Este estándar está dividido en tres capas: la capa del cliente, la capa intermedia y la capa de persistencia.

1. La capa del cliente

La capa del cliente equivaldría a la capa del cliente en la tradicional estructura cliente-servidor. La componen principalmente aplicaciones cliente, applets, clientes Web y otras GUIs; y se ubica en la computadora del cliente.

2. La capa intermedia

La capa intermedia se divide a su vez en otras dos capas: capa de presentación y capa de lógica de negocio.

Bajo esta subdivisión aparece la idea principal del patrón Model-View-Controller, donde se hace posible la separación entre los datos de la aplicación y su presentación.

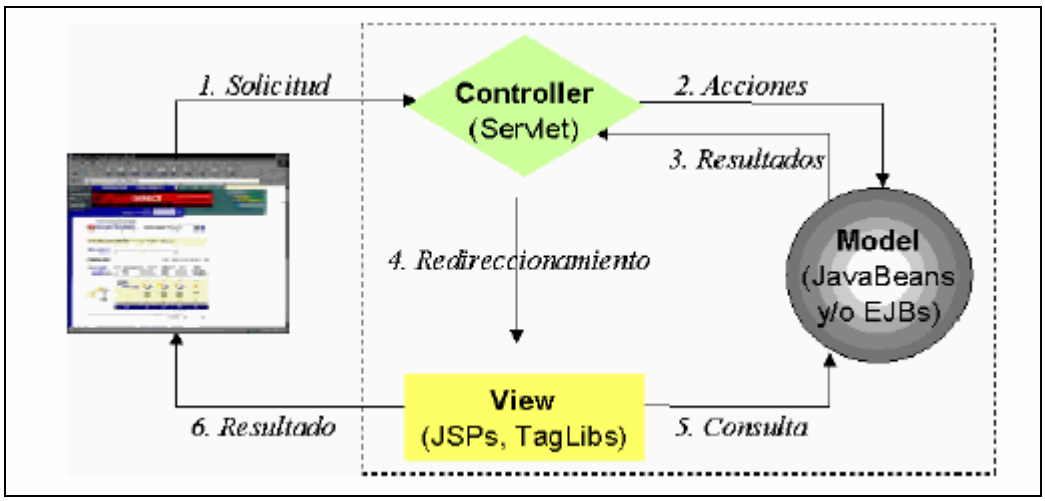


Figura 2: Diagrama del patrón Model-View-Controller

2.1. La capa de presentación

La capa de presentación se encarga de preparar los contenidos web con los que interactúa el usuario, utilizando páginas JSP y Servlets. Para su implementación en GICE4S se ha utilizado el framework Struts, que permite desacoplar la generación de HTML de las acciones a realizar con el modelo.

En esta capa también se han usado presentación HTML para generar la interfaz gráfica de usuario, utilizando páginas JSP, filtros, autenticación de usuarios haciendo uso del protocolo LDAP…

2.2. La capa de lógica de negocio

La capa de negocio encapsula las operaciones sobre el modelo de la aplicación. Esta capa será llamada desde la capa de presentación, y dispone de una interfaz que oculta la tecnología que se ha aplicado en esta capa.

Esta capa está formada por EJBs y otros objetos de negocio.

3. La capa de persistencia

Esta capa engloba los distintos gestores de bases de datos utilizados en la aplicación, incluyendo otros tipos de fuentes de datos como archivos XML o ficheros de texto. Se ubica tanto en el servidor J2EE como en el servidor de base de datos.

C. ONTOLOGÍA

Una parte importante del proyecto ha sido la elaboración de una ontología de la Informática, necesaria para calcular la similitud de las asignaturas. Para ser capaces de elaborarla hemos tenido que familiarizarnos con distintas aplicaciones y tecnologías.

1. Ontology Web Language (OWL)

OWL (Ontology Web Language) ha sido desarrollado por el W3C y tiene como punto de partida las experiencias previas realizadas con DAML-OIL en las cuales se inspiraron los creadores de OWL para desarrollar el lenguaje. OWL es un lenguaje de marcado para la publicación de ontologías en Internet y tiene como objetivo facilitar un modelo de marcado, construido sobre RDF y codificado en XML que permita representar ontologías a partir de un vocabulario más amplio y una sintaxis más fuerte que la que permite RDF. Por este motivo OWL puede ser utilizado para representar de forma explícita el significado de términos pertenecientes a un vocabulario y definir las relaciones que existen entre ellos.

```
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
xmlns:owl="http://www.w3.org/2002/07/owl#" xmlns="http://www.owl-
ontologies.com/unnamed.owl#" xml:base="http://www.owl-
ontologies.com/unnamed.owl">
   <owl:Ontology rdf:about=""/>
   <owl:Class rdf:ID="DL3-Modular_Design_of_Combinational_Circuits">
      <rdfs:subClassOf>
         <owl:Class rdf:ID="DL-Digital_Logic_Topics"/>
      </rdfs:subClassOf>
   </owl:Class>
   <owl:Class rdf:ID="HCI6-Human-centered_Software_Development">
      <rdfs:subClassOf>
         <owl:Class rdf:ID="HCI-Human_Computer_Interaction_Topics"/>
      </rdfs:subClassOf>
   </owl:Class>
   <owl:Class rdf:ID="DL4-Memory_Elements">
      <rdfs:subClassOf>
         <owl:Class rdf:about="#DL-Digital_Logic_Topics"/>
      </rdfs:subClassOf>
   </owl:Class>
   <owl:Class rdf:ID="OS7-File_Systems">
      <rdfs:subClassOf>
         <owl:Class rdf:ID="OS-Operating_Systems_Topics"/>
      </rdfs:subClassOf>
   </owl:Class>
(…)
```

Figura 3: Ejemplo de código OWL

OWL se divide en tres sublenguajes OWL-Lite, OWL-DL y OWL-Full, cada uno de los cuales proporciona un conjunto definido sobre el que trabajar, siendo el más sencillo OWL-Lite y el más completo OWL-Full.

Para la realización de la ontología en OWL hemos utilizado la aplicación de código abierto Protégé 3.0, que se explica en el siguiente apartado.

2. Protégé

Protégé es una aplicación desarrollada por el Stanford Medical Informatics en la Stanford University School of Medicine con apoyo de instituciones militares, médicas, tecnológicas y científicas de los Estados Unidos.

Este programa permite la creación de ontologías de forma visual ocultando el engorroso código en OWL que hay por detrás. Así la creación de clases, individuos y propiedades, que son los elementos básicos en OWL, se hace de forma más sencilla.

A medida que se va construyendo la ontología se puede ir comprobando su corrección porque Protégé permite comprobar la consistencia, clasificar y computar los tipos inferidos mediante el razonador Racer, que se explicará en el siguiente apartado. Además, se puede exportar la ontología construida a Java, HTML, CLIPS, OWL, RDF, etc. Para la realización de este proyecto

utilizamos la ontología en el lenguaje OWL, como ya se ha explicado anteriormente.

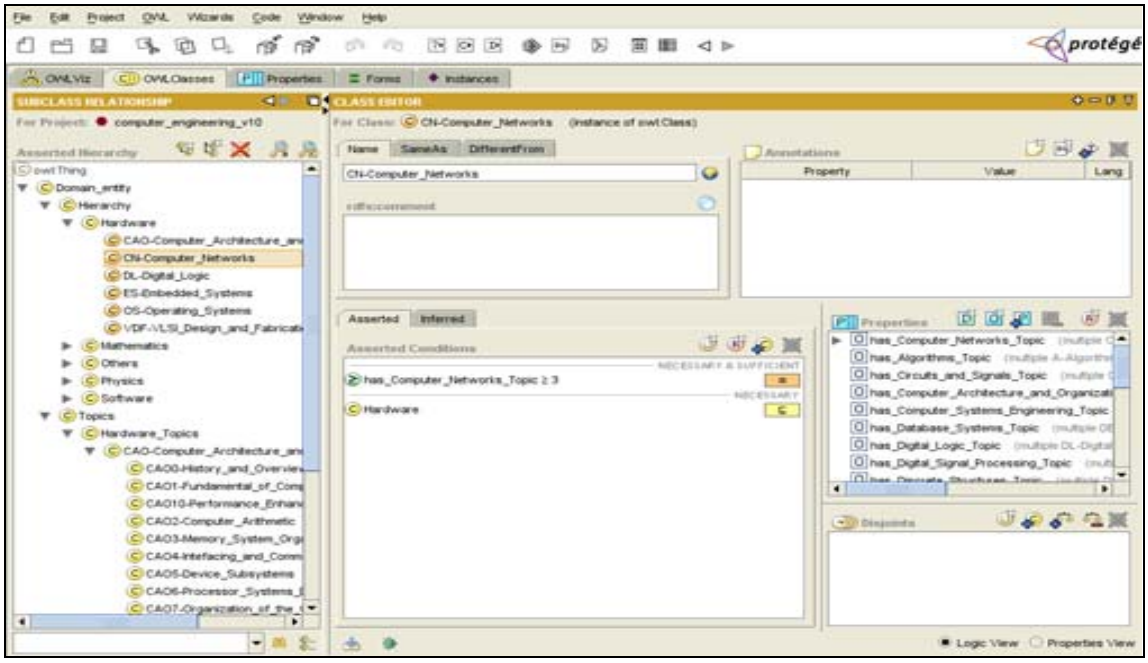


Figura 4: Captura de pantalla de la aplicación Protégé

3. DIG: RACER, FaCT y Pellet

Estos tres sistemas, RACER, FaCT y Pellet, implementan el estándar DIG basado en HTTP para la interconexión de sistemas DL con interfaces y aplicaciones usando un protocolo basado en XML. Además, algunos de ellos tienen un lenguaje de consultas propio.

RACER es un sistema de representación del conocimiento que implementa un cálculo optimizado de tableau para una descripción lógica muy expresiva. Puede razonar sobre varios ABoxes y TBoxes aunque para nuestro proyecto ha sido suficiente con utilizar uno sólo. Ha sido implementado inicialmente por la Universidad de Hamburgo aunque actualmente está soportado por la Concordia University (Montreal) y la University of Applied Sciences (Wedel, Hamburgo).

```
(define-concrete-domain-attribute year :type cardinal)
(define-concrete-domain-attribute days-in-month :type cardinal)

(implies Month (and (>= days-in-month 28) (<= days-in-month 31)))

(equivalent month-inleapyear
       (and Month
              (divisible year 4)
              (or    (not-divisible year 100)
                     (divisible year 400))))
(equivalent February
       (and Month
              (<= days-in-month 29)
              (or    (not month-inleapyear)
                     (= days-in-month 29))
              (or    month-inleapyear
                     (= days-in-month 28))))

(instance feb-2003 February)
(constrained feb-2003 year-1 year)
(constrained feb-2003 days-in-feb-2003 days-in-month)
(constraints (= year-1 2003))

(instance feb-2000 February)
(constrained feb-2000 year-2 year)
(constrained feb-2000 days-in-feb-2000 days-in-month)
(constraints (= year-2 2000))

(constraint-entailed? (<> days-in-feb-2003 29))
(constraint-entailed? (= days-in-feb-2000 29))
```

Figura 5: Ejemplo de código RACER

FaCT (Fast Classification of Terminologies) y Pellet son otros dos sistemas de código abierto, el primero escrito en Lisp y el segundo en Java, con características similares a RACER. En estos sistemas además se permiten consultas RDQL (RDF Data Query Language), por lo que resultan más generales.

El principal motivo por el que hemos decidido usar RACER y no otros es que permite razonar con archivos OWL integrándose en Java muy fácilmente a través del paquete JRacer que implementa las funciones de RACER en Java y sirve como interfaz entre ambas tecnologías o mediante la utilización de Jena, que se verá en el siguiente apartado. Igualmente se podría haber utilizado Pellet con Jena pero desde las etapas anteriores ya habíamos usado RACER, que se integra perfectamente con Protégé.

4. JRacer y Jena

Una vez decidido que se va a usar RACER, queda decidir cómo se va a conectar con nuestras aplicaciones en Java: usando JRacer o Jena.

Jena es un sistema mucho más general que JRacer dado que permite la conexión con los tres razonadores comentados en el apartado anterior y,

además, se pueden hacer consultas utilizando RDQL. Todo esto lo hacen mucho más general pero a la vez más complicado.

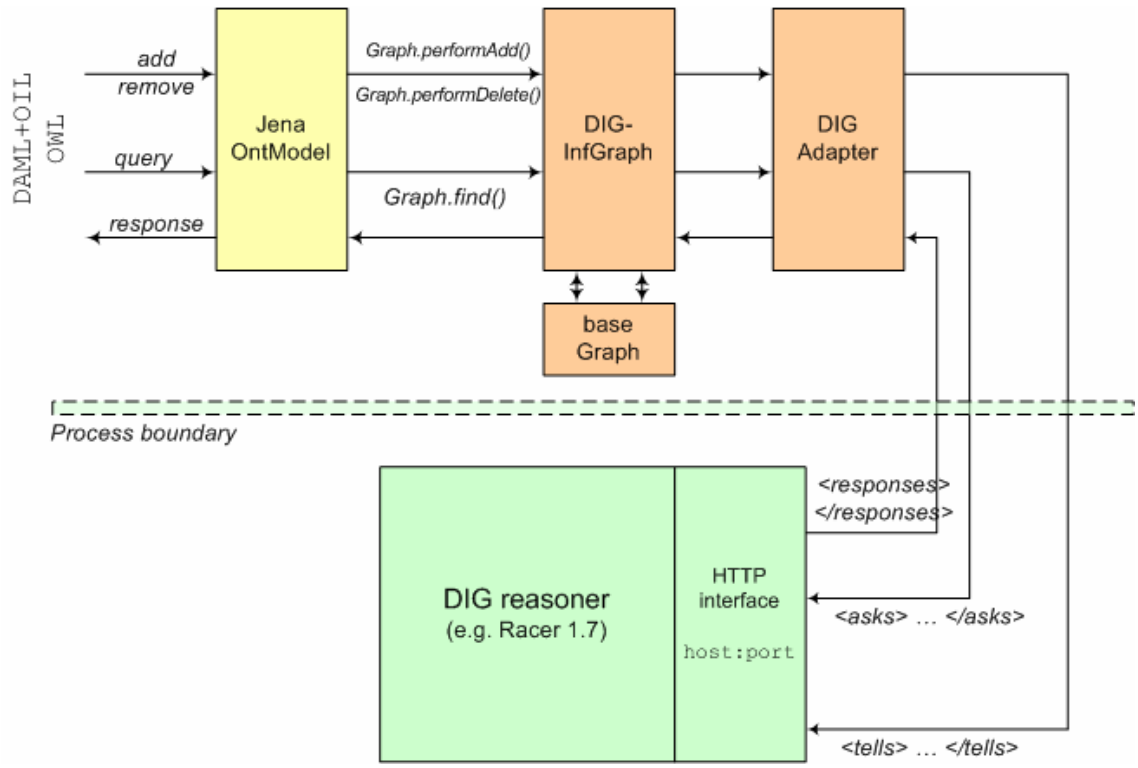


Figura 6: Diagrama de uso de Jena con un razonador DIG

JRacer es específico para usar con RACER y archivos OWL que es precisamente lo que usamos en el proyecto. Por este motivo es más sencillo y hemos decidido usarlo, dado que se ajusta a nuestras necesidades perfectamente.

5. Racer Interactive Client Environment (RICE)

Rice es un entorno que provee a RACER de una interfaz gráfica muy útil para realizar razonamientos y pruebas sin tener que escribir código en Java ni en RACER. Aunque con Protégé se pueden hacer razonamientos con RACER, no permite realizar todas las consultas disponibles, tan sólo las más básicas para comprobar si una ontología es consistente. Sin embargo, Rice nos permite realizar de forma sencilla cualquier consulta, por lo que es una herramienta casi imprescindible para trabajar con RACER.

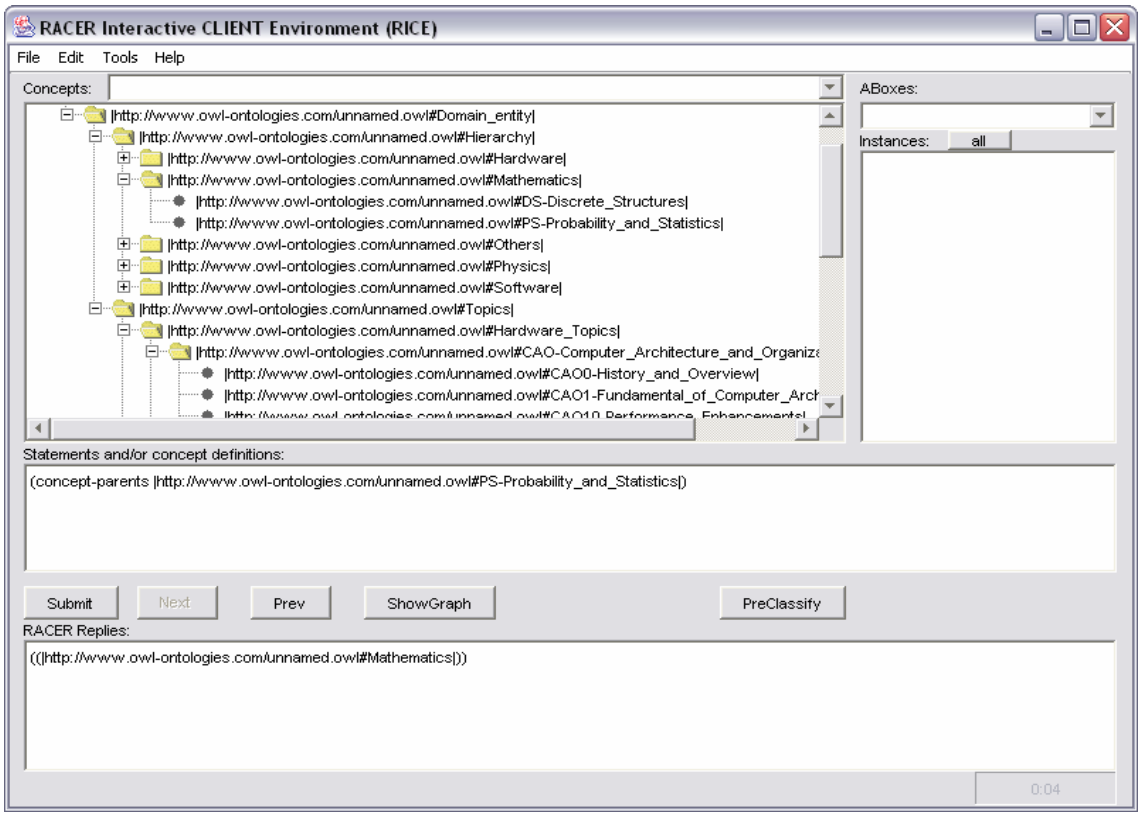


Figura 7: Captura de pantalla de RICE

6. Instance Store (IS)

Instance Store (IS) ha sido desarrollado por la Universidad de Manchester. Es una aplicación Java creada para llevar a cabo un razonamiento eficiente y escalable en lógicas descriptivas sobre individuos.

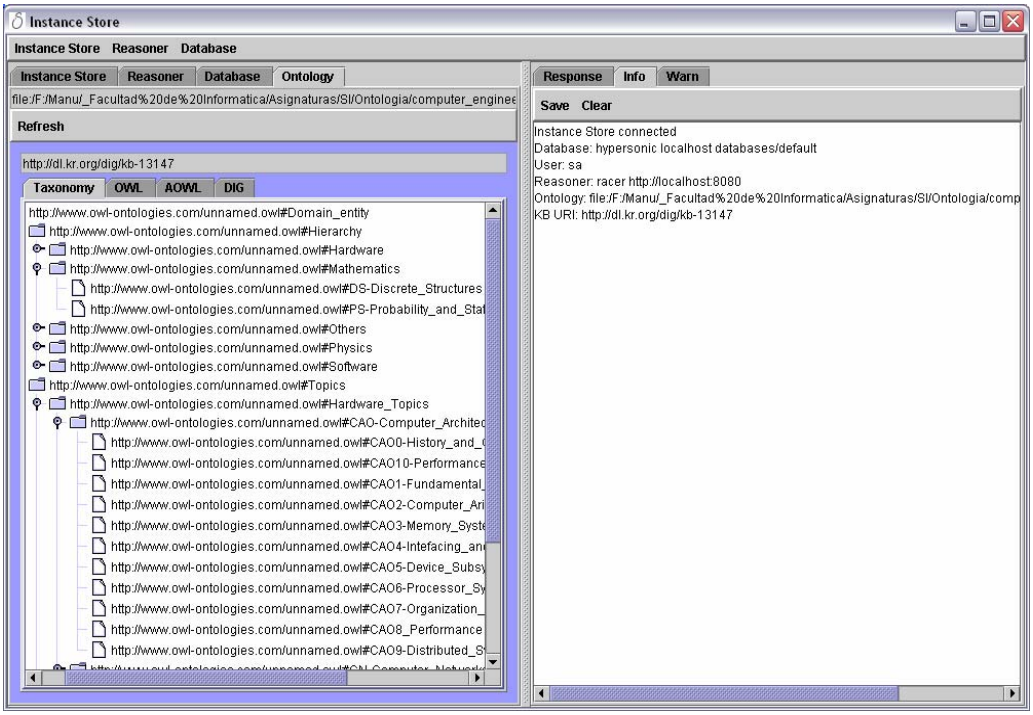


Figura 8: Captura de pantalla de IS

Como se muestra en la arquitectura del sistema en el siguiente diagrama, IS ofrece interconexión entre el razonador y una base de datos lo que permite almacenar en ella los individuos, juntos con la información deducida por el razonador sobre la posición en la taxonomía ontológica de sus descripciones correspondientes.

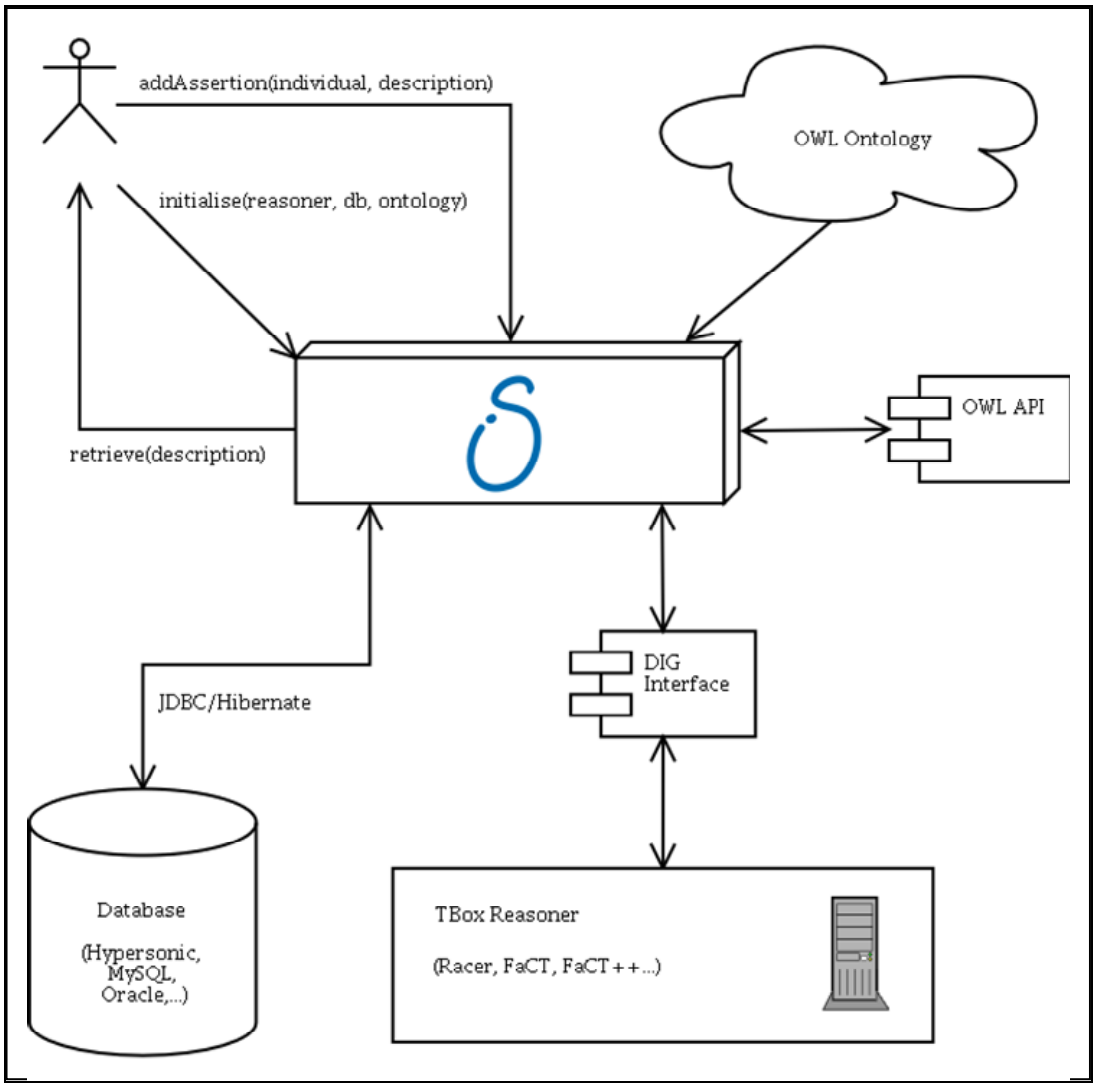


Figura 9: Diagrama de uso de IS

Este sistema puede trabajar con distintas bases de datos como Hypersonic, MySQL u Oracle y con distintos razonadores como Racer o FaCT.

Finalmente descartamos el uso de este sistema para nuestro proyecto porque complica en exceso el diseño al tener que integrarlo en la arquitectura J2EE y no se obtienen grandes ventajas a manejar con código propio el almacenaje de los individuos en la base de datos. La única precaución que hay que tomar es que haya consistencia entre los datos de la base de datos y los de la ontología, esto es, que coincidan los nombres en uno y otro lugar.

D. BASE DE DATOS

Para el desarrollo de nuestro proyecto se ha considerado el uso de distintos sistemas gestores de base de datos. El proyecto GICE4S usa MySQL, que es un sistema gestor de base de datos gratuito que ofrece grandes

prestaciones, por lo que optamos por usar el mismo sistema gestor para no complicar el diseño del proyecto.

E. APLICACIONES

En definitiva hemos tenido que usar numerosas aplicaciones para el desarrollo del módulo de inteligencia artificial para el proyecto GICE4S y para su integración en la arquitectura J2EE del mismo. Entre ellas destacamos:

- El servidor de aplicaciones JOnAS de código abierto desarrollado por Bull y Evidian, que implementa en Java las especificaciones de J2EE. Utilizamos este servidor porque es el que se usa en el proyecto GICE4S.

- Eclipse 3.0 creado por Bull como entorno de desarrollo, porque ofrece una fácil integración con el servidor JOnAS.

- Otros paquetes como Lomboz, Jakarta Struts y Jakarta Taglibs relacionados con la arquitectura J2EE, así como el servicio de directorio LDAP para la autenticación de usuarios.

- MySQL como servidor de bases de datos.

- Protégé, Racer, JRacer y RICE relacionados con el desarrollo de la ontología y cálculo de la similitud entre asignaturas.

CAPÍTULO IV:

FASE DE ADQUISICIÓN DE CONOCIMIENTO

A. INTRODUCCIÓN

Este capítulo está dedicado a la adquisición de conocimiento llevada a cabo durante la primera fase del proyecto. En él se recogen los procesos desarrollados, así como los resultados obtenidos.

Durante esta fase, principalmente, se decidió desarrollar una base de datos para mantener la información estática del proyecto, así como una ontología para almacenar aquella información sobre la cual fuese necesaria realizar algún tipo de razonamiento.

B. ENTREVISTA CON EL COORDINADOR ERASMUS

Después de leer la documentación necesaria acerca de la aplicación GICE4S, nos surgieron una serie de dudas para llevar a cabo la implementación del proyecto. Por esta razón acordamos una entrevista con el coordinador Erasmus de la Facultad de Informática, Pedro González Calero.

Las preguntas que se expusieron, así como las respuestas que se dieron fueron las siguientes:

- ¿Un alumno puede estar matriculado en dos carreras de la misma Universidad o estar matriculado en dos Universidades?

  Sí, existe esa posibilidad, sin embargo no es necesaria contemplarla en el proyecto.

- Cuando se comparan asignaturas, si estas utilizan distintos sistemas de créditos, ¿cómo se realiza la comparación?

  Se hace contabilizando la carga existente en un curso. En nuestro caso, la carga de un curso es de 70 créditos. Para realizar la comparación se establece una relación directamente proporcional entre sus créditos ECTS y los nuestros.

- Cuando se realiza el proceso de convalidación, ¿se tiene en cuenta el carácter (obligatorio, optativo, troncal y libre elección) de la asignatura de la Universidad de destino?

  La equivalencia entre asignaturas, tanto por contenidos como por número de créditos, ha de ser bastante estricta para las asignaturas troncales y obligatorias, relajada para las asignaturas optativas y muy relajada para asignaturas de libre elección.

- ¿Existe algún límite en cuanto al número de créditos en los que hay que matricularse, tanto en la Universidad de origen como en la de destino?

No, sin embargo se recomienda matricularse como mucho de un curso académico que consta de 60 créditos.

- ¿Cómo se traducen las calificaciones de las asignaturas cuando las tablas de equivalencias no coinciden?

  El procedimiento utilizado es la transformación al sistema ECTS, que actualmente es numérico.

- ¿Hay Universidades que puedan impartir las clases en otro idioma además del oficial?

  Existe la posibilidad, sin embargo cada año la oferta suele variar.

Una vez resueltas las dudas planteadas, el coordinador Erasmus añadió los siguientes datos:

- Para solicitar una beca Erasmus es necesario haber aprobado el primer curso académico.

- Se pueden considerar estables los planes de estudio actuales.

C. DESARROLLO DE LA BASE DE DATOS

Para el desarrollo de nuestro proyecto se ha considerado el uso de distintos sistemas gestores de base de datos. El proyecto GICE4S usa MySQL por lo que optamos por usar el mismo sistema gestor para no complicar el diseño del proyecto, aunque la arquitectura J2EE nos ofreciera otras posibilidades.

Después de obtener los datos necesarios en la entrevista anteriormente referida con el coordinador Erasmus, optamos por un diseño de la base de datos que contenía una serie de tablas que creíamos que en un principio eran útiles para el desarrollo del proyecto. Sin embargo muchas de ellas dejaron de ser necesarias según fuimos avanzando en el desarrollo de la aplicación. Entre ellas se encontraban las siguientes:

- Entidades:

  - Degree: Almacena información acerca de la titulación.

  - Curriculum: Contiene información acerca del plan de estudios.

  - Agreement: Contiene información acerca de los descriptores que han de cumplir las diferentes asignaturas.

  - LanguageExam: Contiene la información referente a las notas obtenidas por un alumno en las pruebas de idioma.

  - Preferences: Contiene las preferencias del alumno para elegir el destino de la beca Erasmus.

- Relaciones:

  - HasPending: Relación (Subject – Student), indica las asignaturas que el alumno tiene pendientes.

  - HasPassed: Relación (Subject – Student), indica las asignaturas que el alumno tiene aprobadas.

  - CurriculumBelong: Relación (Subject - Curriculum), indica el plan de estudios al que pertenece una asignatura.

  - HasCurriculum: Relación (Currículo- Degree), indica el plan de estudios que sigue una titulación.

  - IsOf: Relación (Student – Degree), relaciona al alumno con la titulación a la que pertenece.

  - DegreeBelong: Relación (Faculty – Degree), indica las titulaciones que tiene una universidad.

  - FacultyBelong: Relación (University - Faculty), relaciona las facultades con la Universidad a la que pertenecen.

El diseño de la base de datos inicial tenía el siguiente aspecto:

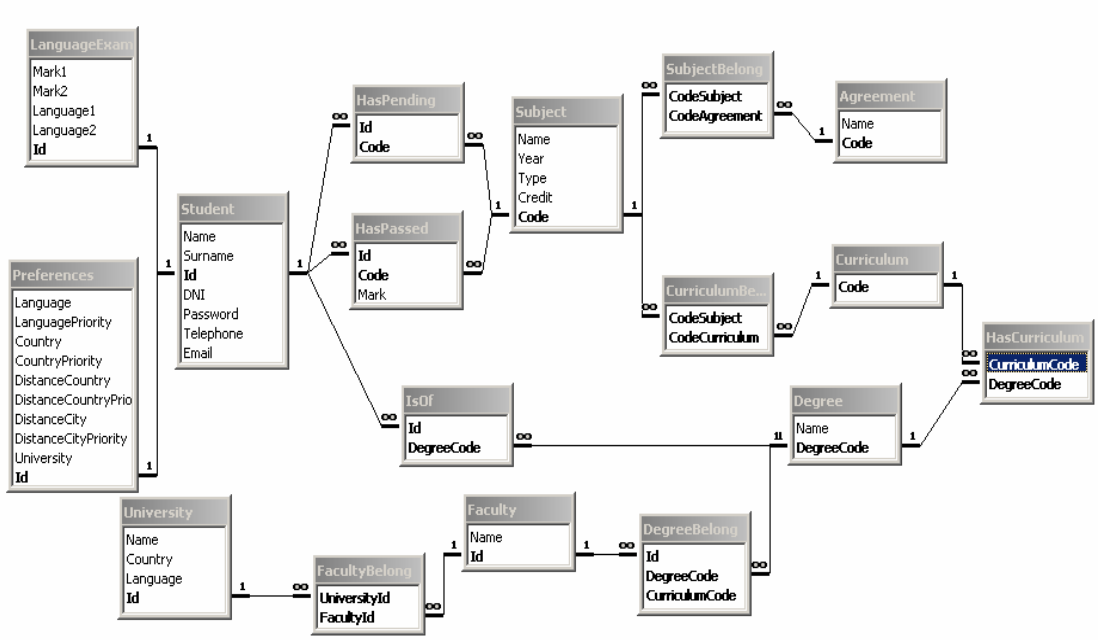


Figura 10: Diagrama de la base de datos inicial

El diseño final de la base de datos que ha sido utilizado durante el desarrollo de la aplicación es el siguiente:

- Entidades:
  - Subject: Contiene información a cerca de las asignaturas.
  - Student: Contiene información a cerca de los datos del alumno.
  - Faculty: Contiene información a cerca de la facultad.
  - University: Contiene información a cerca de la universidad.
  - KnowledgeArea: Almacena las distintas categorías que contiene cada área de conocimiento.
  - Descriptor: Almacena los convenios que contienen las categorías de cada área de conocimiento.
  - Ramas: Contiene las distintas áreas de conocimiento.
  - ConvalidacionOficial: Almacena convalidaciones que ya han sido verificadas por el coordinador Erasmus.
- Relaciones:
  - FacultyBelong: Relación (University - Faculty), indica las facultades que contiene una universidad.

- SubjectBelong: Relación (Subject - Agreement), indica los convenios que engloba cada asignatura.

- AreaBelong: Relación (knowledgeArea - Descriptor), indica los convenios que contiene cada área de conocimiento.

- RamasBelong: Relación (Ramas - knowledgeArea), indica las categorías que contiene cada área de conocimiento.

- DescriptorBelong: Relación (Descriptor - knowledgeArea), indica los convenios que contiene cada categoría.

La mayor parte de los datos introducidos en las tablas provienen de fuentes reales de conocimiento:

- Subject: Los datos han sido extraídos en base a la información encontrada en la página web de la facultad a cerca de las asignaturas que en ella se imparten.

- Student: Datos ficticios.

- Faculty: Los datos almacenados se corresponden con las facultades pertenecientes a las universidades, que en el año 2004-2005 tienen convenio Erasmus con la facultad de Informática.

- University: Los datos almacenados se corresponden con las universidades que en el año 2004-2005 tienen convenio Erasmus con la facultad de Informática.

- KnowledgeArea: Los datos almacenados se corresponden con la información obtenida del documento “Computer Curriculum – Computer Engineering” elaborado por la IEEE Computer Society y ACM.

- Ramas: Los datos han sido obtenidos del documento “Computer Curriculum – Computer Engineering”.

- Descriptor: Al igual que en los casos anteriores, para establecer los convenios de cada categoría se consideró el documento “Computer Curriculum – Computer Engineering”.

El diseño de la base de datos utilizado para el desarrollo final del proyecto tiene el siguiente aspecto:

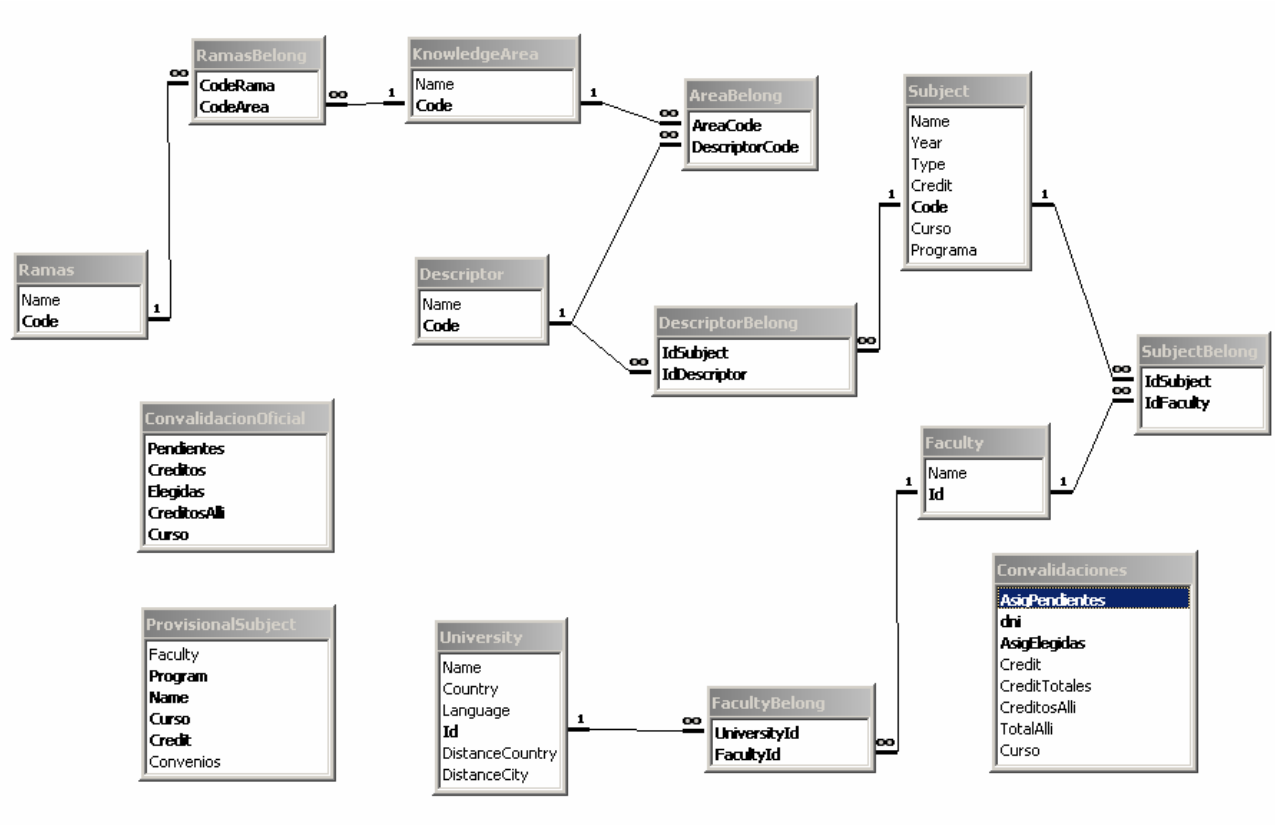


Figura 11: Diagrama de la base de datos del proyecto

D. DESARROLLO DE LA ONTOLOGÍA

Una parte importante del proyecto ha sido la elaboración de una ontología de la Informática, necesaria para calcular la similitud de las asignaturas.

La ontología está formada por cinco áreas de conocimiento que forman la Ingeniería Informática. Cada área de conocimiento está constituida a su vez por distintas categorías que incluyen los convenios que deben englobar cada una de ellas. De esta manera, asignando a cada asignatura una serie de convenios, podemos encuadrarla en una o varias áreas de conocimiento y así clasificarla dentro de la ontología.

Para su desarrollo hemos partimos del documento “Computer Curriculum – Computer Engineering” elaborado por la IEEE computer Society y ACM, que presenta una posible división de la Ingeniería Informática en áreas de conocimiento.

Las distintas áreas de conocimiento en que se divide la ontología son las siguientes:

- Software: Engloba las distintas categorías relacionadas con el área del software.

- Hardware: Engloba las distintas categorías relacionadas con el área del hardware.

- Mathematics: Engloba las distintas categorías relacionadas con el área de las matemáticas.

- Physics: Engloba las distintas categorías relacionadas con el área de la física.

- Others: Otras.

Su estructura está definida detalladamente en el Apéndice A.

La ontología forma parte del módulo de implementación de la función de similitud de asignaturas, por lo que es necesario cargarla en la aplicación cada vez que se utiliza dicha función, así como lanzar RACER para su correcto funcionamiento, como ya se indicó en la fase de investigación. Todo este proceso conllevaría una carga de trabajo considerable cada vez que quisiéramos hacer uso de la similitud de asignaturas, por ello se ha desarrollado una aplicación que genera un archivo que contiene todos los valores de similitud entre los descriptores presentes en la ontología y de esta forma no será necesario utilizar el razonador RACER ni su interfaz para Java JRacer.

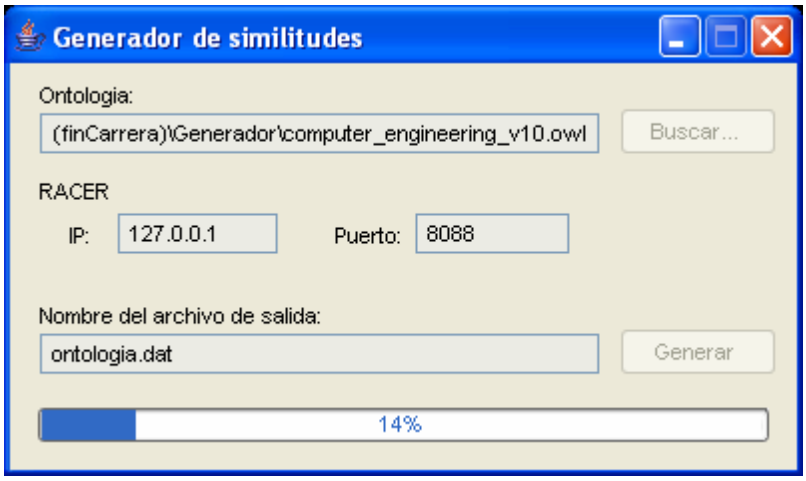


Figura 12: Captura del generador de similitudes

La aplicación recoge en el archivo "ontología.dat" toda la información necesaria para su posterior uso en el cálculo de la similitud de asignaturas.

E. RI (RECUPERADOR DE INFORMACIÓN)

Para el desarrollo del proyecto, es necesario conocer las asignaturas que pueden cursar los alumnos en otras universidades. Además, para que el sistema decida si una signatura es convalidable por otra debemos conocer entre otros datos, los contenidos que abarca. Como ya se explicó en la parte de este capítulo dedicada al desarrollo de la ontología, el sistema utiliza unos descriptores que sería conveniente que su uso se generalizara en todas las Universidades del programa Erasmus, siendo el docente o el coordinador de la asignatura el que determinara cuáles de estos descriptores cubre una asignatura.

Hasta el momento este proceso se realiza manualmente, leyendo el temario de cada asignatura y definiendo los descriptores que cubre cada una. Esto unido a la dificultad que supone el empleo de distintos idiomas en la descripción de los programas de las asignaturas y que la persona dedicada a ello debe ser experta en la materia para poder realizar esta labor de forma adecuada, supone un trabajo muy laborioso.

Para facilitar esta tarea se ha desarrollado un recuperador automático de información, cuya finalidad sería darle el programa de la asignatura y que le asigne los descriptores que cubre la asignatura de forma automática. Este sistema utiliza un razonamiento basado en casos por lo que requiere de una fase de entrenamiento. Esta fase es muy importante y laboriosa; importante porque si no se eligen textos adecuados o poco representativos la aplicación no funcionará correctamente; y laboriosa porque si queremos que sea capaz de clasificar los programas entre casi doscientas categorías habrá que introducir un número elevado de textos de entrenamiento. Por este motivo se ha incluido la posibilidad de introducir los textos mediante el teclado, mediante un fichero o introduciendo la URL de la página Web que contenga el texto.

Para facilitar aún más esta tarea se ha desarrollado un asignador automático de descriptores (AsAuD). Este programa lee de la base de datos del proyecto las direcciones de Internet donde se encuentran los programas de las asignaturas y los descarga. Una vez hecho esto utiliza el clasificador para asignar unos descriptores a la asignatura y almacena el resultado en un archivo SQL.

El recuperador automático se explica detalladamente en el capítulo V dedicado a la fase de desarrollo.

CAPÍTULO V:

FASE DE DESARROLLO

A. RI (RECUPERADOR DE INFORMACIÓN)

1. Introducción

Para el correcto funcionamiento del proyecto, se deben conocer las asignaturas que pueden cursar los alumnos en otras universidades. Esto es evidente. Además, para que el sistema decida si una signatura es convalidable por otra, su similitud en definitiva, debemos conocer los contenidos que abarca entre otras cosas. El sistema utiliza unos descriptores definidos en la ontología (definida en el Apéndice A) y lo ideal sería que estos descriptores (u otros definidos de tal forma que abarquen todo el área de conocimiento) sirvieran como estándar y referente para todas las Universidades del programa Erasmus, definiendo el docente o el coordinador de la asignatura en el programa de la misma cuáles de estos descriptores cubre una asignatura.

Como esto todavía no es así, hay que hacer este proceso manualmente por una persona externa, leyendo el temario de cada asignatura y decidiendo que descriptores cubre con todo el trabajo que esto conlleva además de la dificultad que supone el idioma. Otro problema es que esta persona debe ser un experto en la materia para poder realizar esta labor de forma adecuada.

Para facilitar esta tarea se ha desarrollado un recuperador automático de información, cuya finalidad sería darle el programa de la asignatura y que le asigne los descriptores que cubre la asignatura de forma automática. Este sistema utiliza un razonamiento basado en casos por lo que requiere de una fase de entrenamiento. Esta fase es muy importante y laboriosa; importante porque si no se eligen textos adecuados o poco representativos la aplicación no funcionará correctamente; y laboriosa porque si queremos que sea capaz de clasificar los programas entre casi doscientas categorías habrá que introducir un número elevado de textos de entrenamiento. Por este motivo se ha incluido la posibilidad de introducir los textos mediante el teclado, mediante un fichero o introduciendo la URL de la página Web que contenga el texto.

Para facilitar aún más esta tarea se ha desarrollado un asignador automático de descriptores (AsAuD). Este programa lee de la base de datos del proyecto las direcciones de Internet donde se encuentran los programas de las asignaturas y los descarga. Una vez hecho esto utiliza el clasificador para asignar unos descriptores a la asignatura y almacena el resultado en un archivo SQL.

2. Especificación de requisitos

El sistema clasifica textos dentro de unas categorías definidas por el usuario, sin ningún tipo de restricción en cuanto al ámbito de las mismas. Los textos pueden ser introducidos por el usuario escribiéndolos, cargándolos de un archivo de texto o cargándolos de una página Web. Para que el sistema funcione correctamente los textos introducidos desde Internet deben estar en

HTML puro sin JavaScript ni otros añadidos similares. El sistema sólo considera el texto mostrado por pantalla (sin etiquetas).

En cuanto al número de categorías no se ha puesto ninguna restricción, pero hay que tener en cuenta que para un gran número de categorías hay que poner muchos textos de entrenamiento para que el sistema pueda funcionar correctamente. Muchos textos de entrenamiento hacen que el programa sea más lento como cabe esperar. Se podría decir que el número de textos de entrenamiento para cada categoría debe ser proporcional al número de categorías para un buen funcionamiento. Si aceptamos esto, al aumentar las categorías linealmente, los textos totales de entrenamiento deberán aumentar cuadráticamente.

En nuestro caso se han introducido 187 categorías, una por cada descriptor de la ontología; y para cada categoría se ha introducido un texto representativo de la misma. Posteriormente se podrán añadir más textos de entrenamiento para que el sistema aprenda y se comporte mejor.

3. Conocimiento utilizado

Distinguimos entre dos tipos de conocimiento, el procedimental y la base de casos.

3.1. Base de Casos

El conocimiento utilizado por el sistema es introducido por el usuario. Consiste en un corpus de documentos clasificados que le sirven al sistema como entrenamiento para el aprendizaje. Los documentos son introducidos en forma de archivos de texto o de páginas Web y el sistema hace una copia de ellos en archivos de texto en una ruta determinada, evitando así los posibles problemas que surjan del cambio de ubicación de los documentos o indisponibilidad de la red para acceder a las páginas Web.

Todo este conocimiento se guarda de la siguiente forma:

- Un archivo de texto (.txt) por cada documento de entrenamiento de nombre “<nombre de la categoría>_<número de documento>.txt”

- Un archivo (.train) que almacena el número de categorías y sus nombres, la ubicación de donde se han sacado los documentos de entrenamiento de cada categoría y de las copias realizadas.

El usuario actúa como experto clasificando los documentos en las distintas categorías antes del entrenamiento, en este caso debe ser una persona que conozca las distintas ramas de la informática.

Una vez introducidos todos los documentos del corpus se realiza el entrenamiento, que transforma la representación del conocimiento en otra más manejable para calcular similitudes. Se generan los siguientes archivos:

- Un archivo de datos (.dat) por cada categoría que almacena un vector de palabras y pesos que representan el perfil de la categoría. El archivo tiene como nombre “<nombre de la categoría>.dat”

- Un archivo (.cat) que guarda la información del número de categorías y su nombre, y de la ubicación de los datos. Este archivo será el que luego utilizará el clasificador para cargar los datos de las categorías.

Esta forma de organizar el conocimiento es lo permite tener varios conjuntos de categorías independientes y utilizar el programa para clasificar textos dentro de categorías muy concretas, como en este caso, los 187 descriptores; o bien, clasificar las asignaturas en áreas de conocimiento más generales si se prefiere.

3.2. Conocimiento procedimental

El conocimiento procedimental utilizado por el programa está en el propio código del programa en forma de algoritmo.

El programa clasificador calcula la similitud entre dos casos, un texto a clasificar y una categoría. Esto se lleva a cabo realizando una transformación de ambos en un vector y calculando el coseno del ángulo que forman.

La transformación de una categoría en un vector se realiza en la fase de entrenamiento. Primero se transforma cada documento de entrenamiento en una lista de raíces con la frecuencia de aparición en el texto (quitando las palabras vacías) y posteriormente, se combinan todas las de una categoría (utilizando una lista invertida de índices) para formar un único vector de raíces y pesos que representa el “perfil” de la categoría.

La transformación del texto a comparar se realiza quitando las palabras vacías y extrayendo del resto su raíz. Después se crea el vector de raíces y frecuencias que representa al texto.

Para calcular el coseno primero se transforman los dos vectores para que tengan los mismos ejes. Las raíces que aparezcan en el vector de categorías y no lo hagan en el vector del texto a clasificar se le añaden a este último con peso 0; y las que aparezcan en el vector del texto y no lo hagan en el de categorías se eliminarán del primero. La similitud devuelta estará entre 0 y 1 porque se calcula con el coseno, siendo 1 cuando sean vectores iguales (con el mismo ángulo).

4. Implementación

La aplicación consta de tres programas diferenciados, el entrenador, el clasificador y el asignador automático de descriptores (AsAuD). El entrenador es el encargado de crear los perfiles de las categorías a partir de los datos de entrenamiento introducidos por el usuario; y el clasificador es el encargado de

dar la similitud entre un texto que queramos clasificar y las distintas categorías. El asignador automático de descriptores automatiza y sustituye la labor del clasificador. Si no tuviéramos el asignador automático de descriptores, para cada asignatura de la base de datos, habría que extraer la dirección Web del programa, introducirla en el clasificador, observar los resultados, decidir que descriptores se le asignan a la asignatura e introducirlos en la base de datos. AsAuD realiza esta labor automáticamente dotando de mayor inteligencia artificial al proceso dado que decide qué descriptores, cuántos y de cuánta calidad, asigna a cada asignatura.

Las principales clases de la aplicación son las siguientes:

- EntrenadorGUI: Es la interfaz gráfica del entrenador. Permite guardar y abrir datos de entrenamiento realizados anteriormente además de, como es natural, crear unos nuevos.

- Entrenador: Implementa la lógica del entrenamiento, generando los perfiles de las categorías.

- ClasificadorGUI: Es la interfaz gráfica del entrenador. Permite abrir perfiles de categorías para utilizarlas para clasificar, así como abrir archivos de texto o páginas Web que serán objeto de la clasificación.

- Clasificador: Implementa la lógica del clasificador. Dadas las categorías y el texto a clasificar devuelve una lista ordenada de las categorías y la similitud del texto con cada categoría.

- Transformador: Esta clase es utilizada tanto por el entrenador como por el clasificador. Dado un texto lo transforma en una lista de raíces y frecuencias de aparición de las raíces en el texto.

- GeneradorDescriptores: Implementa la lógica de AsAuD.

Además de estas clases la aplicación consta de otras diez clases menores o auxiliares.

Se entregan todas estas clases junto con el proyecto. Con el proyecto se entregan los siguientes archivos:

- entrenador.exe: el ejecutable del entrenador.

- clasificador.exe: el ejecutable para el clasificador.

- ri.properties: el archivo de propiedades del proyecto.

- stoplist.txt: el archivo que contiene la lista de parada para el inglés.

- listaParada.txt: el archivo que contiene la lista de parada para el español.

- Un directorio “training” en el que el programa buscará por defecto los datos de entrenamiento. Se incluye el entrenamiento realizado para las 187 categorías con los textos utilizados.

- AsAud.exe: el ejecutable del asignador automático de descriptores.

- generador.properties: el archivo de propiedades de AsAuD, donde se podrán configurar la base de datos, el archivo de categorías .cat, los parámetros que deciden cuántos descriptores se asignan, etc.

Antes de usar la aplicación tendremos que configurar el archivo de propiedades para trabajar en inglés o en castellano, en nuestro caso lo más lógico es hacerlo en ingles, dado que la mayoría de las universidades europeas ofreces los programas de las asignaturas en este idioma. Para usarla deberemos crear unas categorías y entrenar al sistema.

Para entrenar al sistema deberemos ejecutar el programa “entrenador.exe” que tiene el siguiente aspecto:

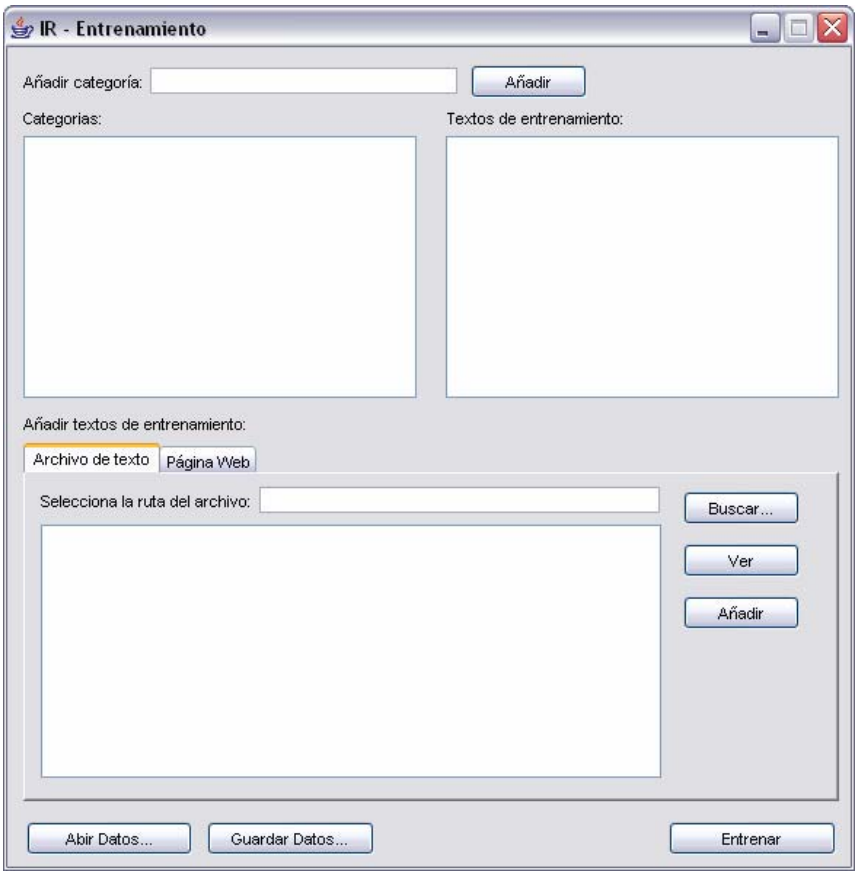


Figura 13: Captura de pantalla del entrenador

A continuación añadiremos las categorías y los textos de entrenamiento. Para añadir una categoría escribiremos el nombre de la categoría y pulsaremos en el botón añadir. Para añadir un texto a una categoría seleccionaremos la

categoría a la que pertenece de la lista de categorías y a continuación podremos seleccionar un archivo de texto o una página Web seleccionando la pestaña correspondiente.

Cuando tengamos todos los documentos de entrenamiento podremos guardarlos para utilizarlos ulteriormente. Para que el programa lleve a cabo el entrenamiento presionaremos el botón entrenar y guardaremos el resultado del entrenamiento en un archivo de extensión “cat”. Este proceso puede llevar más o menos tiempo dependiendo de la cantidad de documentos utilizados en el entrenamiento.

El siguiente paso es ejecutar el clasificador (clasificador.exe), para clasificar un temario en concreto, o bien, ejecutar AsAuD para realizar una asignación de descriptores a todas las asignaturas de la base de datos. El clasificador tiene el siguiente aspecto:

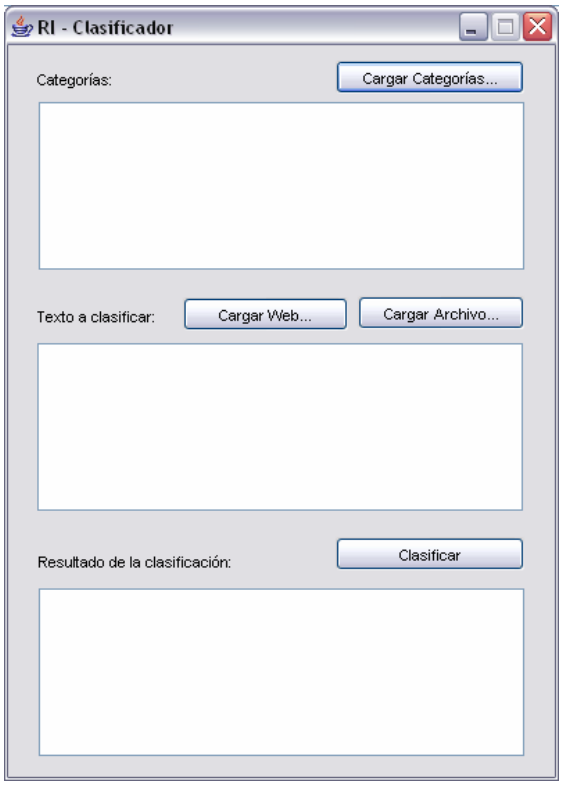


Figura 14: Captura de pantalla del clasificador

Para realizar la clasificación de un archivo deberemos abrir un archivo de extensión cat (el que hemos creado anteriormente u otro que queramos utilizar) y el archivo a clasificar (desde un fichero o desde una Web) y le daremos al botón clasificar. El programa nos devolverá la lista de las categorías ordenadas desde la que más se parece a la que menos y entre paréntesis aparecerá un valor entre 0 y 1 correspondiente con la similitud.

Si decidimos usar AsAuD deberemos configurarlo primero mediante el archivo “generador.properties”. Las propiedades más relevantes que tiene el archivo son:

- categorias: Es la ruta del archivo de categorías con extensión cat generado por el entrenador anteriormente con los datos de los descriptores. Este archivo permite ordenar la lista de descriptores según su similitud con el temario de la asignatura.

- porcentajeCorte: Valor entre 0 y 100 que indica el porcentaje de similitud de corte a partir del cual se considera que se puede asignar un descriptor a una asignatura. Los descriptores que no superen este umbral de similitud no serán asignados a la asignatura.

- minDescriptores y maxDescriptores: Indican cuántos descriptores se asignarán a cada asignatura como mínimo y como máximo respectivamente. El valor mínimo está supeditado a que la similitud de los descriptores pase el porcentaje de corte.

- tiempoEspera: Esta propiedad contiene el número de milisegundos que el programa espera para obtener el programa de cada asignatura de la dirección de Internet de la base de datos.

- archivoSalida: Ruta del archivo SQL en el que se guardará la salida del programa, las consultas para insertar en la base de datos los descriptores asignados automáticamente.

- Existen otras propiedades para configurar la conexión con la base de datos como el nombre, la contraseña, etc.

Una vez configurado tan sólo habrá que ejecutar el archivo AsAuD.exe y el programa nos irá mostrando por la consola del sistema las operaciones que va realizando así como los posibles errores que vaya encontrando como, por ejemplo, que no se pueda establecer la conexión con alguna página Web de las que contienen los temarios de las asignaturas.

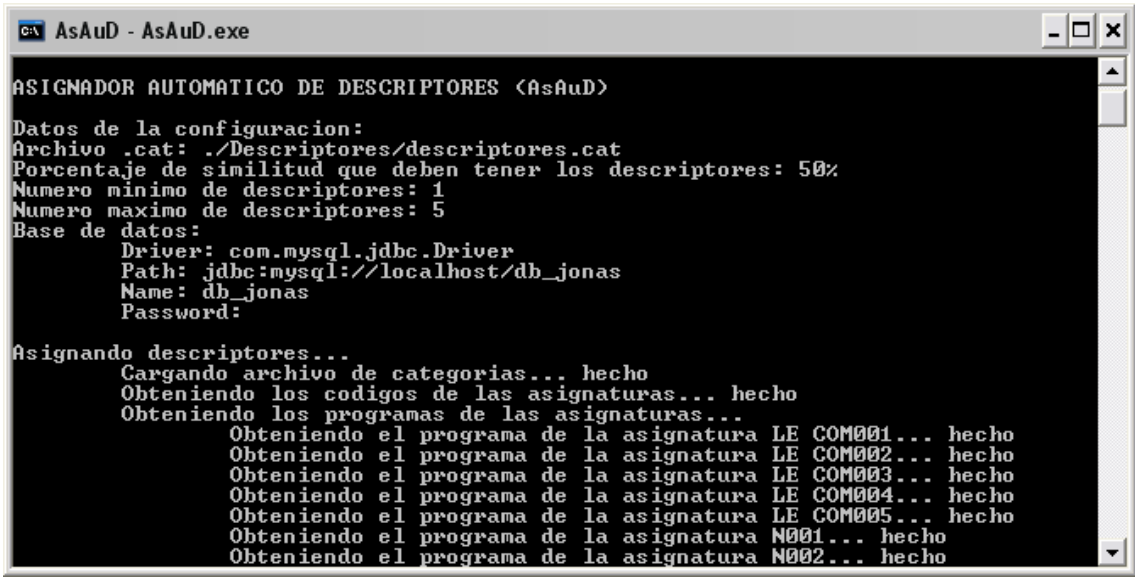


Figura 15: Captura de pantalla de AsAuD

Una vez haya finalizado generará el archivo SQL con las consultas necesarias para introducir los descriptores asignados en la base de datos.

5. Ejemplo del funcionamiento de la aplicación

Para usar el sistema hay que seguir los pasos detallados en el apartado anterior. Una ve creadas las categorías y añadidos los textos de entrenamiento el sistema realiza el entrenamiento propiamente dicho. Para cada texto de cada categoría lo transforma en una lista de raíces con la frecuencia de aparición en el texto. Después, para cada categoría construye una lista invertida de índices y da a cada raíz un peso dependiendo de la frecuencia de aparición en cada texto. Al final para cada categoría se queda con una lista de raíces y pesos que representan las categorías. Esta información es guardada en ficheros y posteriormente es recuperada por el clasificador.

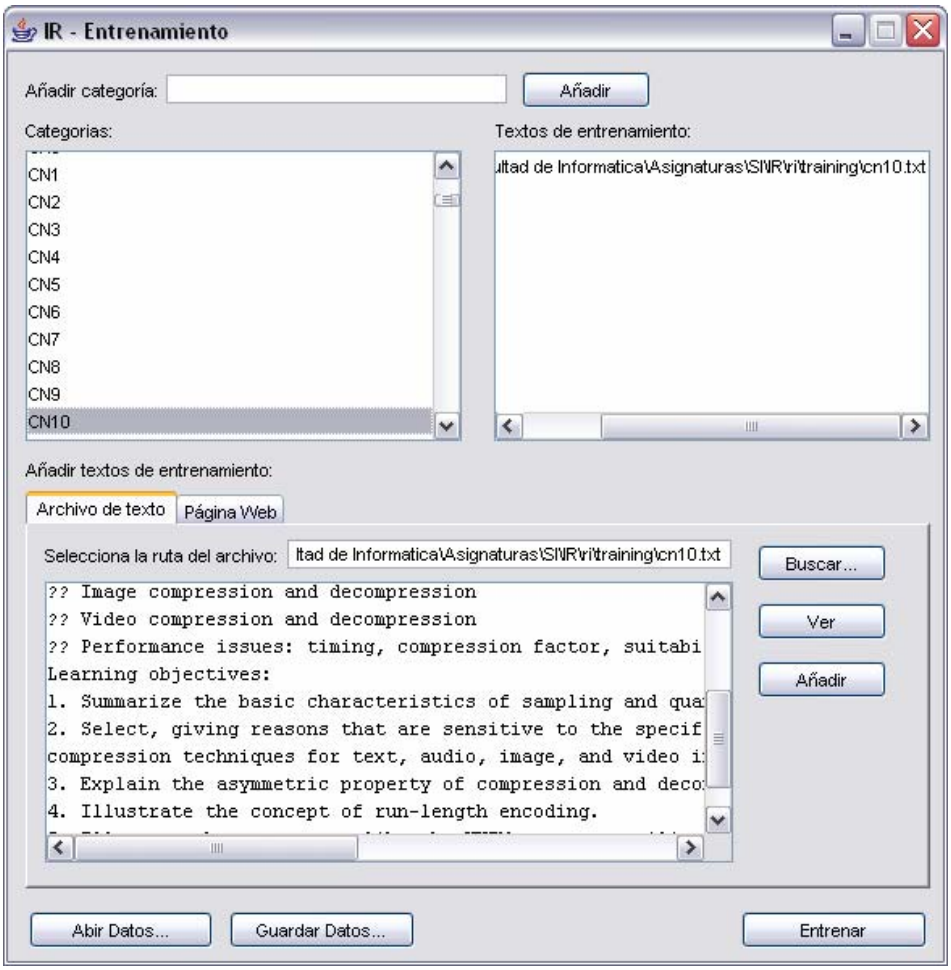


Figura 16: Captura de pantalla del entrenador

Para clasificar se utiliza el clasificador; se cargan las categorías, en nuestro caso los 187 descriptores, y el archivo que se quiere clasificar, un temario de una asignatura colgado en la página Web de una universidad, por ejemplo la asignatura “Software Engineering” de la Universidad de Nicosia (http://www.intercollege.ac.cy/include/custom/PopUpCourseDescription.cfm?CourseId=COMP-453), y se le ordena la sistema que clasifique. El sistema transforma el texto que queremos clasificar en una lista de raíces y frecuencias, tal como se hizo para el entrenamiento. Una vez obtenida, se calcula la similitud con cada categoría y se devuelve la lista de las categorías ordenada con esta similitud. Para calcular la similitud del texto con cada categoría calcula la similitud entre la lista de raíces y frecuencias del texto con la lista de raíces y pesos de cada categoría que había creado el entrenador y que se había almacenado en disco y recuperado por el clasificador posteriormente. Ambas listas son transformadas para contener las mismas raíces y ser tratadas como vectores. Una vez hecho esto se calcula el coseno de ambos vectores, lo que pasará a ser la similitud.

Figura 17: Captura de pantalla del clasificador

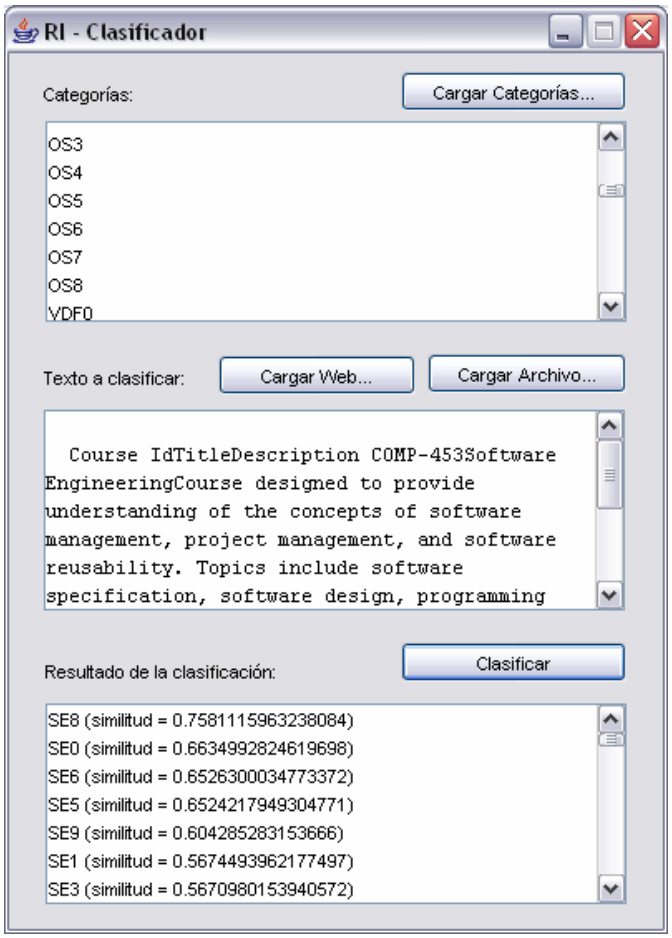


Figura 18: Captura de pantalla del clasificador

Como se puede observar en la figura anterior los descriptores con los que se obtiene una similitud más alta son los relacionados con la Ingeniería del Software, como cabía esperar.

Con el clasificador obtenemos la lista de los descriptores con la similitud y es el usuario el que decide cuántos y cuáles de estos se le asignarán a la asignatura clasificada. Si usamos AsAuD este proceso es realizado automáticamente considerando los criterios establecidos por el usuario. Para este ejemplo en concreto si se establece que se cojan los cuatro descriptores mejores (poniendo minDescriptores = 1, por ejemplo, y maxDescriptores = 4 con un porcentaje de corte muy bajo) el programa asignará a la asignatura los cuatro primeros (SE8, SE0, SE6, SE5); y, sin embargo, si decidimos asignar los descriptores que pasen de un umbral de similitud de 0.6, por ejemplo, (poniendo porcentajeCorte = 60 y maxDescriptores con un valor muy alto) el programa asignará los cinco primeros descriptores (los cuatro anteriores y SE9).

6. Validación y verificación

Para la verificación y la validación del programa se han realizado numerosas pruebas con distintos ejemplos de distintos ámbitos y hemos comprado que tiene un funcionamiento correcto y que mejora sensiblemente cuantos más y mejores textos de entrenamiento haya.

El problema de asignar descriptores a asignaturas resulta especialmente complejo por el gran número de éstos que consideramos. Como se ha podido ver en el ejemplo anterior se ha obtenido un muy buen resultado pero al haber puesto solamente un texto por cada categoría esto no siempre es así. La solución es añadir más textos de entrenamiento en cada categoría consiguiendo que el programa sea más fiable.

7. Conclusiones

Como se ha comentado en el apartado anterior, el programa funciona correctamente pero necesita de una gran cantidad de textos de entrenamiento, tarea que resulta muy laboriosa pero que es imprescindible para que el sistema sea lo más fiable posible.

Mientras que no se alcance un grado de fiabilidad muy elevado, la aplicación debe ser considerada como una ayuda o un asistente a la tarea de dar descriptores a las asignaturas de otras universidades europeas, que se deberá realizar a mano pero de una forma mucho más sencilla porque consistirá en revisar lo propuesto por el sistema. Otra alternativa es contar con la ayuda de los estudiantes Erasmus que han terminado su periodo en el extranjero para que asignen ellos descriptores a las asignaturas que han cursado, obteniendo así textos de entrenamiento para aumentar el corpus.

En cualquier caso, y aunque el sistema sea muy fiable porque se ha hecho un esfuerzo por entrenarlo adecuadamente, sería conveniente que se

comprobaran los resultados obtenidos por la gran trascendencia y repercusión que tendría dentro del proyecto una asignación de descriptores mal realizada.

Hay que tener en cuenta que este programa ha sido desarrollado para el inglés y que no todas las universidades del programa Erasmus tienen los temarios de las asignaturas en inglés. Si en un futuro se quiere profundizar por esta vía, una posible solución es utilizar Eurowordnet como se explica en el Apéndice B, pero esto ya queda fuera de los objetivos del proyecto.

B. SIMILITUD ENTRE ASIGNATURAS

En este apartado vamos a tratar uno de los aspectos más importantes del proyecto, el cálculo de la similitud entre dos asignaturas, que será imprescindible para realizar las funcionalidades del módulo de inteligencia artificial del proyecto GICE4S.

Una vez que sepamos calcular cuánto se parecen dos asignaturas cualesquiera podremos aplicar este conocimiento a multitud de utilidades muy prácticas para el sistema. En el proyecto se ha usado este cálculo de la similitud en la implementación del sistema de ayuda al alumno para la elección de universidad, en el sistema de ayuda al alumno para la elección de asignaturas y en el sistema de ayuda al coordinador Erasmus para ver si una asignatura es convalidable. Todos estos sistemas se explican en detalle en los siguientes apartados de este capítulo.

Cuando a un estudiante le conceden una beca Erasmus cursará en el extranjero una serie de asignaturas que luego convalidará a su regreso en su facultad de origen. Las convalidaciones de estas asignaturas se suelen hacer de una a una, es decir, una asignatura de su facultad se le convalida por una asignatura cursada en el extranjero. Aunque este sea el caso general, también es perfectamente posible que se produzcan convalidaciones de varias a varias, esto es que se convaliden varias asignaturas de su facultad por una cursada en el extranjero o viceversa, que se convalide una asignatura por varias cursadas en el extranjero. Por este motivo se ha desarrollado una función de similitud que realiza el cálculo de una a una y otra que realiza en cálculo de varias a varias.

1. Una a una

1.1. Introducción

En este apartado se va a tratar el cálculo de la similitud una a una, es decir, cuánto se parecen dos asignaturas. Para la realización de este cálculo se tendrán en cuenta varios aspectos de las mismas como los créditos que tienen, el carácter de las mismas y, por supuesto, el contenido.

A todos los efectos consideraremos el cálculo de la similitud como el cálculo de cómo de convalidable es una asignatura, es decir, cuanto más se acerque a uno este valor más convalidable será la asignatura y cuánto más se acerque a cero menos convalidable será. Podemos hacer esta consideración porque utilizamos los mismos criterios en el cálculo de la similitud que en la toma de la decisión de si una asignatura es convalidable.

Esto último provoca que la función de similitud que vamos a construir no sea simétrica, al contrario de lo que suele ser habitual en las funciones de similitud en otros ámbitos. Una asignatura A y B pueden tener un valor de similitud alto si consideramos que A es la asignatura en la facultad de origen y

B es la asignatura en la facultad de destino y B cubre más temario que A y además tiene más créditos; por supuesto, A sería convalidable por B. Pero si B fuera la asignatura de la facultad de origen y A la de la facultad de destino no serían convalidables puesto que A no cubre todo el temario de B y además tiene menos créditos. Por lo tanto en este último caso el valor de similitud deberá ser menor.

$$similitud(A, B) \neq similitud(B, A)$$

1.2. Especificación de requisitos

Para calcular la similitud entre dos asignaturas será necesario conocerlas completamente con carácter previo a la realización del cálculo. Conocemos completamente una asignatura cuando sabemos los datos necesarios para atacar el problema que estamos tratando; estos son los créditos que tiene la asignatura, el carácter de la misma y, lo más importante, los descriptores que tiene asignados. El problema de la asignación de descriptores se expone en el apartado anterior de este mismo capítulo.

Cuando se conocen los datos enumerados anteriormente deben formar un objeto de la clase `Asignatura`, que es con la que trabaja la función. El resultado devuelto por la misma es un `double` con un valor que va comprendido entre 0 y 1.

1.3. Conocimiento utilizado

El conocimiento utilizado para el cálculo de la similitud es básicamente ontológico. Este conocimiento es adquirido en la fase de adquisición del conocimiento realizada previamente, como se ha explicado en el capítulo IV, y da lugar a la creación de la ontología comentada en ese mismo capítulo y en el apéndice A.

1.4. Implementación

Como se ha comentado en la especificación de requisitos, los datos principales que debemos conocer para calcular la similitud de una asignatura son el contenido y la duración de la asignatura. El contenido de una asignatura lo vamos a representar como una lista de descriptores. Estos descriptores serán algunos de los casi dos centenares descritos en la ontología y que se pueden consultar en el apéndice A. Por otro lado, la duración de la asignatura se medirá en créditos.

Sabiendo esto podemos considerar la similitud entre dos asignaturas como la combinación de dos similitudes parciales: la similitud por contenido y la similitud por créditos.

1.4.1. Similitud por contenido

Para calcular la similitud por contenido se comparan las listas de convenios de las asignaturas. Dos asignaturas tendrán un valor de similitud por contenido igual a uno en el caso de que tengan los mismos descriptores. Esta similitud irá disminuyendo a medida que los descriptores van siendo más diferentes. La similitud de los descriptores, que se comentará más adelante, utiliza el razonador RACER a través de su interfaz con Java JRacer.

Lo primero que hacemos es contar el número de descriptores que coinciden en ambas listas y calculamos una nueva similitud entre las listas obtenidas de las anteriores descartando los descriptores iguales. La similitud será una combinación del número de descriptores iguales que hayamos encontrado y la similitud de los descriptores que no son iguales. Tras varias pruebas hemos llegado a la conclusión de que la mejor forma de combinar estos valores es mediante una media ponderada, dando más peso al número de descriptores iguales.

$$A = \{d_1, d_2, d_3, \cdots, d_k, a_1, a_2, a_3, \cdots, a_n\}$$
$$B = \{d_1, d_2, d_3, \cdots, d_k, b_1, b_2, b_3, \cdots, b_m\}$$

$$numConveniosIguales = k, \quad a_i \neq b_j \forall i, j$$

$$similitudContenido = \left(\frac{k}{k+n} \times 0'8\right) + \left(similitudConvenios\left(\bigcup_{i=1}^{n} a_i, \bigcup_{j=1}^{m} b_j\right) \times 0'2\right)$$

Para calcular la similitud de dos conjuntos de convenios en los que no hay ninguno igual calculamos la similitud de cada convenio de un conjunto con cada uno del otro conjunto, obteniendo así una tabla de similitudes de convenios. Lo que queremos conseguir es emparejar los convenios de forma que se consiga la máxima similitud posible en la suma de las similitudes de las parejas, que es lo que más se acerca a la realidad y lo que nos dice verdaderamente cuánto se parecen los conjuntos de convenios. Para lograr esto construimos la tabla antes mencionada y buscamos el máximo valor de similitud en ella. La pareja que dé lugar a este valor de la similitud será seleccionada, quitando por tanto la fila y la columna correspondientes a los convenios seleccionados. Repetimos este proceso hasta que la tabla quede vacía consiguiendo así las parejas óptimas.

$$s_{ij} = similitudConvenio(a_i, b_j)$$

$$\begin{bmatrix} s_{11} & \cdots & s_{1q} & \cdots & s_{1m} \\ \vdots & \ddots & \vdots & \cdots & \vdots \\ s_{p1} & \cdots & s_{pq} & \cdots & s_{pm} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ s_{n1} & s_{n2} & s_{nq} & \cdots & s_{nm} \end{bmatrix} \quad \max_{i=1,j=1}^{n,m}(s_{ij}) = s_{pq} \quad \xrightarrow{\text{quitando } p \text{ y } q} \quad \begin{bmatrix} s_{11} & \cdots & \cdots & \cdots & s_{1m} \\ \vdots & \ddots & & \cdots & \vdots \\ \vdots & & \ddots & & \vdots \\ \vdots & \vdots & & \ddots & \vdots \\ s_{n1} & s_{n2} & \cdots & \cdots & s_{nm} \end{bmatrix}$$

Para calcular la similitud entre dos convenios hacemos uso de la ontología. Los descriptores están clasificados en la misma en una jerarquía según a las áreas de conocimiento a las que pertenecen. Aprovechando esta característica y utilizando en razonador RACER a través de su interfaz para Java JRacer calculamos cual es el primer antecesor común. Cuánto más próximo esté más similares serán; por el contrario cuanto más cerca de la raíz esté este antecesor común serán más distintos.

El problema de utilizar RACER es que el cálculo de todas estas similitudes se hace muy lento por lo que se ha optado por guardar todas las similitudes posibles entre convenios en fichero de datos para que la función sea más eficiente. Este fichero de texto se ha creado a través de un programa que invoca al razonador, tal y como se ha explicado en el apartado D del capítulo IV.

1.4.2. Similitud por créditos

Para calcular la similitud por créditos hay que tener en cuenta que esta función tampoco debe ser simétrica. En el caso de que los créditos de la asignatura de la facultad de destino sean mayores o iguales que los de la asignatura de la facultad origen, la similitud debe ser 1 porque esto quiere decir que el alumno ha dado más horas de esa asignatura que las que hubiera dado en su facultad de origen por lo que en este aspecto la asignatura sería convalidable y se le da el máximo valor de similitud.

En el caso contrario, en el que los créditos de la asignatura de la facultad destino sean menores le daremos un valor de similitud acorde la diferencia de créditos. Es decir, que cuanta más diferencia haya entre los créditos de una y de otra la similitud ha de ser menor y cuanto más se parezcan la similitud ha de ser mayor. Además, hay que tener en cuenta que el valor de similitud asignado ha de estar entre 0 y 1. Para conseguir esto hay varias posibilidades pero la que más refleja la realidad es el cociente entre ambos valores de los créditos.

$$similitudCreditos = \frac{creditosDestino}{creditosOrigen} \in (0,1)$$

1.5. Conclusiones

El desarrollo de esta función abre un abanico de posibilidades para la creación de sistemas de ayuda al alumno y al coordinador Erasmus que pueden resultar de gran utilidad. Permitirá indicar de forma automática si dos asignaturas son convalidables, evitando el trabajo manual que se hacía hasta el momento de comprobar los temarios y compararlos, además de los créditos de cada asignatura.

Además, se consigue que los criterios por los que se considera una asignatura convalidable ahora sean más precisos y no estén sujetos a la subjetividad de una persona, aunque al final la última palabra la tenga esta última.

2. Varias a varias

2.1. Introducción

Como se ha explicado anteriormente el cálculo de la similitud de una a una es necesario pero también lo es de varias a varias por la posibilidad de convalidar varias asignaturas por varias asignaturas.

Para calcular la similitud de varias a varias utilizamos la similitud una a una con las asignaturas y se nos plantea el problema de cómo combinar las similitudes parciales de forma que el resultado sea una similitud que se ajuste a la realidad.

Además, esta función tampoco será simétrica por el mismo motivo que no lo era la similitud una a una.

2.2. Especificación de requisitos

Para que se lleve a cabo el cálculo de la similitud de varios a varios sin ningún tipo de problema las asignaturas deben cumplir los mismos requisitos que para el cálculo de la similitud una a una, con la salvedad que ahora la función no va a considerar objetos de la clase `Asignatura` sino *arrays* de objetos de la clase `Asignatura`.

2.3. Conocimiento utilizado

El conocimiento utilizado para el desarrollo de esta función incluye el conocimiento que se utilizó para el desarrollo de la función de similitud de una a una; incluyendo, además, la propia función de similitud de una a una como conocimiento procedimental.

2.4. Implementación

Para la implementación de esta función se consideraron varias posibilidades utilizando la función de similitud creada con anterioridad. Para explicarlas primero es preciso conocer cuales son los datos de entrada y cuál el resultado que queremos obtener.

$$similitudAsignaturas(asignaturasOrigen, asignaturasDestino) \in [0,1]$$

$$asignaturasOrigen = \{A_1, A_2, A_3, \cdots, A_n\} \quad A_i \in Asignatura$$

$$asignaturasDestino = \{B_1, B_2, B_3, \cdots, B_m\} \quad B_j \in Asignatura$$

Una de las opciones manejadas ha sido la de calcular el valor de la similitud para todas las combinaciones posibles de asignaturas y luego combinarlas mediante alguna función de agregación como la media o el producto.

$$s_{ij} = similitudAsignatura(A_i, B_j)$$

Utilizando la media tendríamos:

$$similitudAsignaturas = \frac{\sum_{i=1}^{n}\sum_{j=1}^{m} s_{ij}}{n \times m}$$

Utilizando el producto tendríamos:

$$similitudAsignaturas = \prod_{i=1}^{n}\prod_{j=1}^{m} s_{ij}$$

Pero todas estas aproximaciones y las que utilizan las combinaciones de los valores de la función de similitud de todas con todas no reflejan lo que entendemos por similitud, como cuánto de convalidable es una asignatura. Esto se debe a que se están comparando contenidos de las asignaturas de origen en todas las de destino cuando en realidad lo que nos interesa es que esos contenidos estén en alguna de ellas y no en todas.

Por este motivo optamos por otra forma de calcular la similitud. En lugar de calcular todas las similitudes parciales y luego combinarlas decidimos combinar primero las asignaturas y luego calcular la similitud.

$$similitudAsignaturas(asignaturasOrigen, asignaturasDestino) \in [0,1]$$

$$asignaturasOrigen = \{A_1, A_2, A_3, \cdots, A_n\} \quad A_i \in Asignatura$$

$$asignaturasDestino = \{B_1, B_2, B_3, \cdots, B_m\} \quad B_j \in Asignatura$$

Cada asignatura está formada por un conjunto de descriptores y el número de créditos:

$$Asignatura = \{ (\{d_1, d_2, d_3, \cdots, d_l\}, nc), \quad d_k \in Descriptor, \quad nc \in \mathrm{N} \} = (ld, nc)$$

Combinamos cada lista de asignaturas creando una nueva asignatura en la que su conjunto de descriptores es la unión de todos descriptores de las asignaturas de la lista, y el número de créditos es la suma de los créditos de cada asignatura.

$$\{A_1, A_2, A_3, \cdots, A_n\} \quad A_i \in Asignatura \xrightarrow{combinando} A = \left(\bigcup_{i=1}^{n} ld_i, \sum_{i=1}^{n} nc_i \right)$$

$$\{B_1, B_2, B_3, \cdots, B_m\} \quad B_j \in Asignatura \xrightarrow{combinando} B = \left(\bigcup_{j=1}^{m} ld_j, \sum_{j=1}^{m} nc_j \right)$$

$$similitudAsignaturas(\{A_1, A_2, A_3, \cdots, A_n\}, \{B_1, B_2, B_3, \cdots, B_m\}) = similitudAsignatura(A, B)$$

Con las nuevas asignaturas obtenidas como combinación de las listas de asignaturas origen y destino, invocamos a la función de similitud de una a una que ya habíamos implementado obteniendo así un valor que se ajusta más a la realidad.

2.5. Conclusiones

Esta función resulta de gran utilidad para comprobar si se pueden convalidar asignaturas de varias a varias de forma automática. Hacerlo manualmente para estos casos resulta, además de más engorroso, más complicado, aunque en general no estén implicadas muchas asignaturas.

Además, también se consigue que los criterios por los que se considera unas asignaturas convalidables por otras sean más precisos y objetivos, fuera de la subjetividad de una persona.

C. SISTEMA DE AYUDA AL ALUMNO PARA LA ELECCIÓN DE LA UNIVERSIDAD DESTINO

1. Introducción

Como se mencionó en la introducción, este apartado es una de las líneas principales de inteligencia de las cuales se pretende dotar al sistema.

Puede ser considerado como una parte importante del desarrollo ya que supone una ayuda fundamental para los alumnos, facilitándoles la realización de actividades que hasta el momento se hacían de forma manual y requerían un tiempo notable para su desarrollo.

A modo de ejemplo, basta considerar no solo lo tedioso que puede ser para el alumno el buscar las asignaturas dentro de la universidad destino, sino también el tener que comprobar cada uno de los temarios ofertados para valorar el grado de similitud que presentan con respecto a la asignatura que se desea cursar en la universidad de origen. Este proceso supondría evaluar el contenido de los mismos y los créditos que tienen asignados.

Con el empleo del sistema esta labor queda reducida a unos simples minutos.

2. Especificación de requisitos

Para que un alumno pueda emplear este asistente como ayuda para la elección de universidad, es preciso que previamente se haya dado de alta en el sistema. Este es un requisito indispensable, ya que aquellos alumnos no registrados no tienen acceso al mismo.

En cuanto a requisitos del sistema, éste se apoya, para su correcto funcionamiento, en una base de datos que ha sido confeccionada atendiendo a las necesidades del mismo.

Para que el sistema funcione correctamente dicha base de datos debe contener una relación de las universidades de destino y origen, y las asignaturas que en ellas se imparten, ya que sin esta información el sistema quedaría inservible. Dicho de otra forma, la base de datos debe estar bien formada y completa, en cuanto a asignaturas y universidades se refiere.

3. Conocimiento utilizado

El conocimiento empleado por este sistema de ayuda al alumno para la elección de la universidad de destino se apoya fundamentalmente en la función que establece la similitud entre asignaturas, explicada en el apartado B de este mismo capítulo.

Aunque esta es la fuente de conocimiento empleada más importante en esta parte del desarrollo, cabe mencionar también el uso de la ontología sobre descriptores del área de la informática, tratada en el capitulo IV y en el apéndice A, pudiéndose consultar estos puntos para más información.

4. Implementación

De forma global, esta funcionalidad podría verse dividida en dos fases. Una primera en la cual se valoran las preferencias que el alumno introduce a través del sistema, y en segundo lugar se considera el nivel de convalidación que tengan las asignaturas que el usuario selecciona, de entre las que se ofrecen en la universidad de origen, y las ofertadas en el lugar de destino.

Con estos valores el sistema devolvería los resultados generados en el proceso de razonamiento, y los mostraría por pantalla ordenados en función de la puntuación obtenida y lo buena que sea la misma.

A continuación se detallarán cada una de las áreas mencionadas con el fin de precisar las implementaciones realizadas.

4.1. Primera fase: Preferencias del alumno

4.1.1. Desarrollo de la interfaz

Se pretende elaborar una interfaz sencilla para el usuario donde pueda seleccionar sus preferencias, incluyendo una explicación sobre los pasos a seguir y los resultados que se originarían.

La interfaz desarrollada es la siguiente:

Figura 19: Captura de pantalla del asistente para elegir Universidad

Se pueden observar los siguientes componentes:

- **Elección de idioma**: Selección del idioma, a elegir de entre todos los ofertados por las universidades destino (alemán, inglés, francés, italiano y portugués).

- **Elección de país**: Selección del país destino donde se encuentran las universidades (Alemania, Austria, Bélgica, Francia, Chipre, Finlandia, Inglaterra, Holanda, Italia, Polonia y Portugal).

- **Elección de la distancia al país**: Selección de la distancia que hay entre los países de origen y destino. (Cerca, lejos y muy lejos).

  Para esta parte se han considerado las distancias tomando como origen Madrid, ya que es el lugar donde se encuentra la Facultad de Informática de la Universidad Complutense.

  En cuanto a los destinos, se ha tenido en cuenta la ciudad donde se encuentra la universidad.

  Para la realización de estos cálculos se ha empleado un sistema que calcula la distancia existente entre dos ciudades del mundo, que puede consultarse en la dirección Web http://www.tutiempo.net/tutiempo.php?pagina=distancias#

- **Elección de la distancia entre el campus y la ciudad**: Selección de la distancia existente entre la ciudad de destino y el campus de la universidad.

Para el cálculo de estos valores se han ido consultando las distintas páginas Web de las universidades de destino, y buscando donde se encontraban situadas con respecto a la ciudad.

Las direcciones consultadas pueden verse en el apartado de bibliografía de este documento.

A cada una de estas cuestiones le acompaña la selección de una valoración de la misma, dicho de otro modo, lo importante que es para el alumno la elección que ha marcado en cada uno de los campos anteriores. Con esto logramos acercarnos más a las preferencias reales del alumno y ofrecemos al sistema la posibilidad de realizar una búsqueda no tan exacta en aquellas cuestiones que no dispongan de una relevancia imprescindible, con lo que el rango de resultados obtenidos en la búsqueda crece enormemente.

Como valoraciones posibles disponemos del siguiente rango, ordenado de menor a mayor relevancia:

- **No relevante**
- **Preferible**
- **Relevante**
- **Imprescindible**

Atendiendo a los extremos del intervalo, si el usuario selecciona la opción “Imprescindible” el sistema realiza una búsqueda exacta de aquellas condiciones a las que acompañe, mientras que el caso opuesto, selección de “No relevante”, implica mirar todas las opciones de valores a las que se encuentre ligado, ya que supone que el usuario no tiene ninguna preferencia al respecto.

En el caso del idioma, se eliminó la opción “Imprescindible”, ya que un porcentaje muy elevado de alumnos desean aprender inglés y este idioma es ofertado en muy pocas universidades, con lo que el filtrado de la base de datos quedaría reducido a un número pequeño de universidades. Eliminando dicha opción se deja al sistema más libertad de elección cuando el idioma seleccionado es la lengua inglesa.

Con los valores intermedios, “Preferible” y “Relevante”, el sistema busca en función de una serie de pesos asignados a cada uno de estos valores.

4.1.2. Implementación de la primera fase

Las acciones a realizar en la aplicación quedan recogidas en el archivo `struts-config.xml`, donde se indican los pasos a seguir en cada una de ellas, así como las clases implicadas.

Para entender el funcionamiento de esta parte de la implementación, se seguirán la secuencia de operaciones que queda reflejada en el archivo de configuración anterior.

A parte de las clases y archivos que se detallan a continuación, el sistema se apoya en una serie de clases auxiliares (`Eleccion`, `EntradaBD`, `Inteligencia` y `Similitud`) recogidas en un paquete llamado `Preferencias`, que permite seleccionar las universidades que mejor se adaptan a las preferencias del alumno.

De acuerdo al orden de aparición en el archivo de configuración mencionado las clases, formularios y JSPs implicados serían:

1. Clase → `InsertParamPreferenciasFormAction.java`

Esta clase se emplea para añadir al formulario `PreferenciasForm` los valores de los campos sobre los cuales seleccionará el alumno sus preferencias. Dichos valores se encuentran almacenados en la base de datos, por lo que esta clase realiza un acceso a la misma para recuperarlos.

Las clases devuelven un mensaje (`mapping.findForward`) que sirve como sincronización entre los distintos elementos. En este caso, se devuelve un aviso de forma que el siguiente elemento a utilizar es el JSP `Preferencias`.

Cada archivo JSP que se construye representa una plantilla, que se rellena con los valores fijos de la cabecera (`header.jsp`) y el pie de página (`footer.jsp`), y con un cuerpo central que varía dependiendo de cada JSP. Estos cuerpos tienen el mismo nombre que la plantilla que los llama terminados con la palabra *Content*. En concreto, en este caso tenemos *`PreferenciasContent.jsp`*.

2. Formulario → `PreferenciasForm.java`

Formulario empleado para almacenar los valores introducidos desde la interfaz en esta primera fase, que contiene simplemente atributos y métodos de acceso.

3. JSP → `Preferencias.jsp`

Este JSP se corresponde con la captura expuesta, y como puede observarse, ofrece la posibilidad de introducir las preferencias al usuario.

4. Clase → `PreferenciasAction.java`

En este punto se hace uso del paquete de clases auxiliares.

Esta clase se encarga de recoger los valores introducidos en el formulario, y transformarlos en objetos de la clase `Eleccion`.

Se construye un objeto por cada una de las elecciones que realiza el usuario (idioma, país, distancia al país y distancia a la ciudad), que contienen el campo de la elección realizada y el valor de la misma. Estos objetos son añadidos a una lista de elecciones para ser tratados posteriormente.

El objetivo final de esta parte es conseguir una lista de las universidades que mejor se adaptan a las elecciones recogidas en la lista anteriormente construida. Se emplea la clase `Inteligencia`.

Primeramente se obtendrán la lista de las posibles universidades que cumplan los requisitos indicados por el usuario. Para ello se realiza un filtrado exacto de la base de datos con aquellos valores que el alumno haya marcado como "imprescindibles", reduciendo así notablemente el dominio de la búsqueda. Estas universidades están representadas como objetos de tipo `EntradaBD` de forma que solo seleccionamos de cada una de ellas los campos que nos interesan para realizara las siguientes comprobaciones.

A cada una de las universidades devueltas se le asignará una valoración calculada en base a la similitud que presente con respecto a los valores que el alumno ha establecido en cada campo del formulario.

Para ello se dispone de funciones que calculan la puntuación para cada uno de los campos.

Para las puntuaciones del "idioma" y "el país de destino" se apoya en una nueva clase `Similitud` que establece unas tablas de similitudes dos a dos entre los idiomas y países, respectivamente.

Estos valores quedan comprendidos entre 0 y 1. A modo de ejemplo la similitud entre Alemania e Inglaterra quedaría representada de la forma `[Alemania][Inglaterra]=0.6`.

Para el caso de las distancias se hace uso de funciones matemáticas.

Una vez calculadas las valoraciones para cada uno de los campos, se suman y se le asignan a las universidades, las cuales son insertadas en una lista ordenada.

5. Clase → `RecogerAction.java`

En ella se recogen de la base de datos las asignaturas que se ofertan en la universidad de origen, para que el alumno seleccione aquellas que desearía que se le convalidasen.

Dichas asignaturas serán mostradas a través del JSP `Asignaturas`.

6. JSP → `Asignaturas.jsp`

Este JSP muestra por pantalla el listado de las asignaturas recogidas a través de la clase anterior, de entre las cuales realizará el alumno su elección. Con los valores recogidos se pasa a la segunda fase, donde se comprobará la similitud entre asignaturas.

El código desglosado puede verse al final del documento.

4.2. Segunda fase: Requisitos de convalidación

4.2.1. Desarrollo de la interfaz

Esta parte del desarrollo pretende estudiar el grado de compatibilidad existente entre asignaturas.

Para ello se presenta al alumno una lista de las asignaturas ofertadas en su universidad de origen, de las cuales deberá seleccionar aquellas que desee le sean convalidadas tras la finalización de la beca Erasmus.

También se le da la posibilidad al usuario de no seleccionar ninguna de ellas, en cuyo caso el sistema mostrará las universidades exclusivamente en función de las preferencias indicadas en fase anterior.

Con la elección del alumno el sistema buscará de entre las universidades que ya han sido seleccionadas en la primera fase, aquellas que tengan una mayor similitud entre las asignaturas en función del número de créditos y los descriptores de las mismas.

Figura 20: Captura de pantalla del asistente para elegir Universidad

4.2.2. Implementación de la segunda fase

De la misma manera que se hizo en el apartado anterior, se mostrarán las clases y JSP empleados, ordenados según su aparición de tal forma que quede más claro el funcionamiento.

1. JSP → `Asignaturas.jsp`

Esta fase parte del JSP mencionado como final de la primera. Tras la selección de asignaturas los valores son recogidos en el formulario `selectedSubjectForm`.

2. Clase → `selectedSubjectForm.java`

Formulario empleado para almacenar los valores introducidos desde la interfaz en esta segunda fase.

3. Clase → `MostrarPreferenciasAction.java`

Esta clase recoge los valores del formulario, necesarios para realizar el cálculo de similitudes.

Es aquí donde se emplea la parte inteligente del sistema, cuyo núcleo central en la función de similitud explicada en el apartado 5b de esta documentación.

En concreto esta clase crea un objeto de tipo `SimilitudAsignaturas`, que se encarga de abrir el fichero de la ontología, soporte de gran importancia en el cálculo de similitudes, y que se explica en el capítulo IV de esta memoria.

El objetivo es otorgar una puntuación a cada una de las universidades que han sido seleccionadas como mejores en la primera fase, de forma que se establezca un ranking entre ellas en función de lo similares que sean sus asignaturas con respecto a las seleccionadas por el alumno. La valoración final de la universidad estará formada por una combinación entre los puntos obtenidos en función de las preferencias que el alumno introdujo en la primera fase, y la puntuación obtenida por la similitud de asignaturas. La forma de obtenerlo es realizando una ponderación entre las dos, de la forma `puntuacionFinal = (puntosSimilitud * 0.5)+(puntosPref * 0.5).`

Para conseguir esta última puntuación, la idea es recorrer la lista de las universidades con el fin de ir extrayendo de ellas todas las asignaturas que se imparten (consultas a la base de datos), e irlas comparando con las que ha seleccionado el alumno a través de la interfaz.

Dicha comparación se establece a través del objeto creado anteriormente, empleando su función de similitud entre dos asignaturas. El cálculo de la similitud devuelve un valor comprendido entre 0 y 1.

Para formar estas puntuaciones se dispone de una variable *maxSim* que almacena la máxima similitud que se obtiene una vez se ha comparado una asignatura de la universidad de destino con todas las que ha seleccionado el alumno. De esta forma la puntuación de la universidad estará compuesta por la suma de los valores máximos de similitudes obtenidos (*maxSim*) de cada asignatura que en ella se imparte.

Al realizarlo de esta forma, se debe tener en cuenta el número de asignaturas que tiene cada universidad, ya que sino podrían obtener una mayor puntuación aquellas que dispongan de más asignaturas aunque sus similitudes sean mayores. Es por ello por lo que se divide la puntuación obtenida entre el número máximo de asignaturas que se imparten.

Finalmente se asignan las puntuaciones totales a cada universidad para que puedan ser mostradas por orden en el JSP `listadoPreferencias.jsp`.

El código desglosado puede verse al final del documento.

4.3. Presentación de resultados

La presentación de resultados (`listadoPreferencias.jsp`), tiene el siguiente aspecto:

Figura 21: Captura de la pantalla de resultados del asistente para elegir Universidad

Se muestra al usuario las universidades ordenadas por puntuaciones, que realmente son la media entre las preferencias que introduce por pantalla y la similitud de asignaturas. Para proporcionar una mayor información se desglosa la puntuación final, en sus dos componentes:

- P.Pref: Puntuación obtenida en base a las preferencias del alumno (sobre 100)

- P.Asig: Puntuación obtenida en base a la similitud de las asignaturas (sobre 100)

Al mostrarlo de esta forma el alumno puede decidir que universidad le conviene más en base a las asignaturas y a sus preferencias.

En el caso en que el alumno no haya introducido asignaturas en la segunda fase, el sistema mostrará el ranking, colocando en el lugar P.Asig los símbolos "-", entendiéndose que las asignaturas no han influido en la decisión.

5. Ejemplo del funcionamiento del sistema

A modo de aclaración se presenta un ejemplo del funcionamiento del sistema.

Supongamos que somos un alumno registrado y nos gustaría disfrutar de una beca Erasmus para aprender francés, y a ser posible cerca de España.

Es importante para nosotros que la universidad se encuentre en la ciudad, ya que no queremos perder demasiado tiempo en desplazamientos.

Con estos datos, rellenaríamos el formulario tal y como aparece en la Figura1., es decir:

- *Idioma*: Seleccionamos "francés" y lo marcamos como "Preferible".

- *País*: Seleccionamos "Francia" y lo marcamos como "No relevante", ya que no nos importa el destino, mientras aprendamos francés.

- *Distancia al país*: Dejamos la que aparece por defecto, "Cerca (<1000km)", ya que para nosotros es relevante si el país se encuentra lejos a cerca, "Relevante".

- *Distancia campus-ciudad*: Queremos poco desplazamiento, por lo marcamos la opción "Menos de 10 km", y lo seleccionamos como "Imprescindible".

Tras completarlo enviamos los datos, pulsando sobre "Continuar".

El siguiente paso sería seleccionar aquellas asignaturas que deseamos cursar en el extranjero. Esto corresponde a la Figura2., de la segunda fase.

Supongamos que queremos que se nos convalide la asignatura de Estadística tras la vuelta. La marcamos y pulsamos sobre "Continuar".

Los resultados obtenidos, son los representados en la Figura3.

Puede observarse que la universidad que mejor se adapta a nuestras exigencias es la universidad de Bordeaux, en Francia, seguida de la de Liège en Bélgica.

Las universidades de París quedan más abajo en la clasificación debido a que aunque se habla francés como idioma, carecen de asignaturas que puedan ser convalidadas con Estadística.

Estas son superadas por las de Italia, ya que aunque no se aprende el idioma seleccionado tiene asignaturas que se asemejan a las que hemos elegido.

6. Validación y Verificación

Para la validación y verificación se han realizado numerosas pruebas y se ha llegado a la conclusión de que el sistema tiene un comportamiento adecuado.

Las primeras pruebas se realizaron empleando una base de datos reducida en la cual se cargaban datos ficticios introducidos por nosotros, de tal manera que se conocía el resultado esperado. Se introdujeron numerosas combinaciones posibles hasta que se pudo concluir que realmente el sistema funcionaba como debía.

El motivo por el cual se realizaron pruebas pequeñas y con datos preparados es debido al tamaño de la base de datos real, ya que contiene numerosas asignaturas tanto de la universidad de origen como de las de destino, de forma que sería muy complicado comprobar todos los cálculos de similitudes entre las posibles combinaciones.

La complejidad del sistema crece a medida que se introducen nuevas asignaturas en las universidades.

7. Conclusiones

Tras la realización de esta parte del desarrollo y observar su funcionamiento, se concluye que presenta una gran utilidad.

En la introducción de este mismo punto, ya se mencionó las ventajas que presenta el disponer de un sistema automático para el desarrollo de esta labor, en contraposición con la elaboración manual existente hasta el momento.

Supone no solo un gran ahorro de tiempo, sino también ofrecer una manera más sencilla y cómoda de realizar la búsqueda de la universidad de destino. Una muestra de ello queda reflejada mediante el ejemplo explicativo del punto 5.

Como ya se mencionó en el apartado de requisitos, el sistema se apoya en una base de datos que contiene las asignaturas ofertadas, la cual debe ser bastante completa.

En esta primera versión del proyecto GICE4S, se cuenta con una base de datos que carece de la suficiente información como para que el sistema elabore resultados realmente correctos, debido a que no existen asignaturas para todas las universidades, y los descriptores de las mismas no son del todo fiables ya que, como se comento en el apartado de “recuperación de información”, tenemos el problema del idioma a la hora de extraerlos de sus respectivos programas.

Una vez se solvente este problema con la base de datos, a medida que ésta vaya creciendo, el sistema presentará una ayuda indiscutible para aquellos alumnos que no tengan claro cual ha de ser su destino al solicitar una beca de este tipo.

D. SISTEMA DE AYUDA AL ALUMNO PARA LA ELECCIÓN DE ASIGNATURAS A CURSAR

1. Introducción

Los alumnos beneficiarios de una beca Erasmus, una vez que les ha sido asignada una universidad de destino, tenían que buscar ellos las asignaturas que iban a cursar buscando los temarios en Internet y proponiendo al coordinador departamental, que es el responsable de dar su aprobación, una serie de asignaturas para convalidar por otras de su universidad de origen. Esta propuesta puede sufrir varias iteraciones hasta llegar a un acuerdo entre las dos partes. Esta gestión de reconocimiento académico le sirve al alumno como garantía de que cuando vuelva de su estancia le serán convalidadas las asignaturas que apruebe.

El objetivo de este sistema es ayudar en esta tarea al coordinador departamental y, sobre todo, al alumno, que es el principal interesado. El alumno indicará que asignatura desea convalidar y se le mostrará la lista de asignaturas de la universidad de destino indicando cuáles de ellas serían convalidables, además se le mostrará la similitud de cada asignatura con la de origen.

Este sistema también les ofrece ciertas garantías de que sus asignaturas serán convalidadas a la vuelta de su estancia en el extranjero, porque utiliza la misma función de similitud que el sistema que ayuda al coordinador Erasmus a decidir si una asignatura es convalidable; aunque siempre tiene la decisión final este último.

2. Especificación de requisitos

Para que un usuario pueda acceder a este sistema de ayuda tiene que estar dado de alta en GICE4S como alumno, además de estar identificado después de poner su nombre de usuario y contraseña.

La elección de asignaturas está pensada para la fase en la que el alumno ya tiene asignada una universidad de destino y tiene que decidir las asignaturas que cursará en su estancia en el extranjero. Por lo tanto, ya sabe que asignaturas desea convalidar con una gran certeza.

En cuanto a la base de datos, debe contener las asignaturas de la facultad de origen, en este caso la Facultad de Informática de la U.C.M., que desea convalidar. Estas asignaturas deben contener los descriptores asignados para que se pueda realizar la comparación con las asignaturas de la facultad de destino, que también deberán estar en la base de datos en las mismas condiciones que las de la facultad de origen.

Estos últimos requisitos se cumplen para la universidad de origen pero no para todas las universidades de destino. Se ha utilizado el programa AsAuD, explicado en el apartado A de este mismo capítulo, para asignar descriptores a las asignaturas de las facultades extranjeras. Por lo tanto, las limitaciones en cuanto al idioma y los formatos en los que se presentan los programas de las asignaturas que imponía esta aplicación determinan que no se encuentren disponibles los descriptores de las asignaturas de algunas facultades. Este problema es fácilmente solucionable en los próximos años con la ayuda de los alumnos que han vuelto de Erasmus de las facultades que presentan esta dificultad, porque con su experiencia pueden realizar una asignación de descriptores para las asignaturas que han cursado.

3. Conocimiento utilizado

El conocimiento utilizado para este sistema de ayuda al alumno para la elección de asignaturas lo podemos dividir por un lado en el conocimiento ontológico del área de la Informática que posee el sistema, tratado en el capítulo IV y en el apéndice A, y por otro lado, en el conocimiento procedimental expresado en forma de función de similitud, explicada en el apartado B de este mismo capítulo, y que es una parte importante de este sistema.

4. Implementación

En este apartado se explicarán detalles más técnicos del sistema y de su implementación, que está dividida en tres etapas: la elección de la facultad, en la que el alumno deberá elegir la facultad en la que quiere que se busquen asignaturas similares; la elección de la asignatura que quiere convalidar, en la que el alumno verá por pantalla asignaturas de su facultad y seleccionará la que quiera convalidar; y, por último, la elección de la convalidación, en la que una vez mostradas las diferentes opciones que tiene el alumno para realizar la convalidación, selecciona una de ellas.

En los siguientes subapartados se tratarán de forma específica cada una de las etapas mencionadas anteriormente.

4.1. Elección de la facultad

4.1.1. Desarrollo de la interfaz

En esta etapa se le pide al alumno que seleccione la facultad con la que quiere realizar las convalidaciones. Una vez que el alumno tenga asignada una facultad de destino deberá realizar las convalidaciones con asignaturas de esa facultad, aunque podrá comprobar similitudes con asignaturas de otras facultades, realizando de nuevo el proceso.

Para elegir la facultad, primero se le pide al alumno que seleccione la Universidad a la que pertenece la facultad.

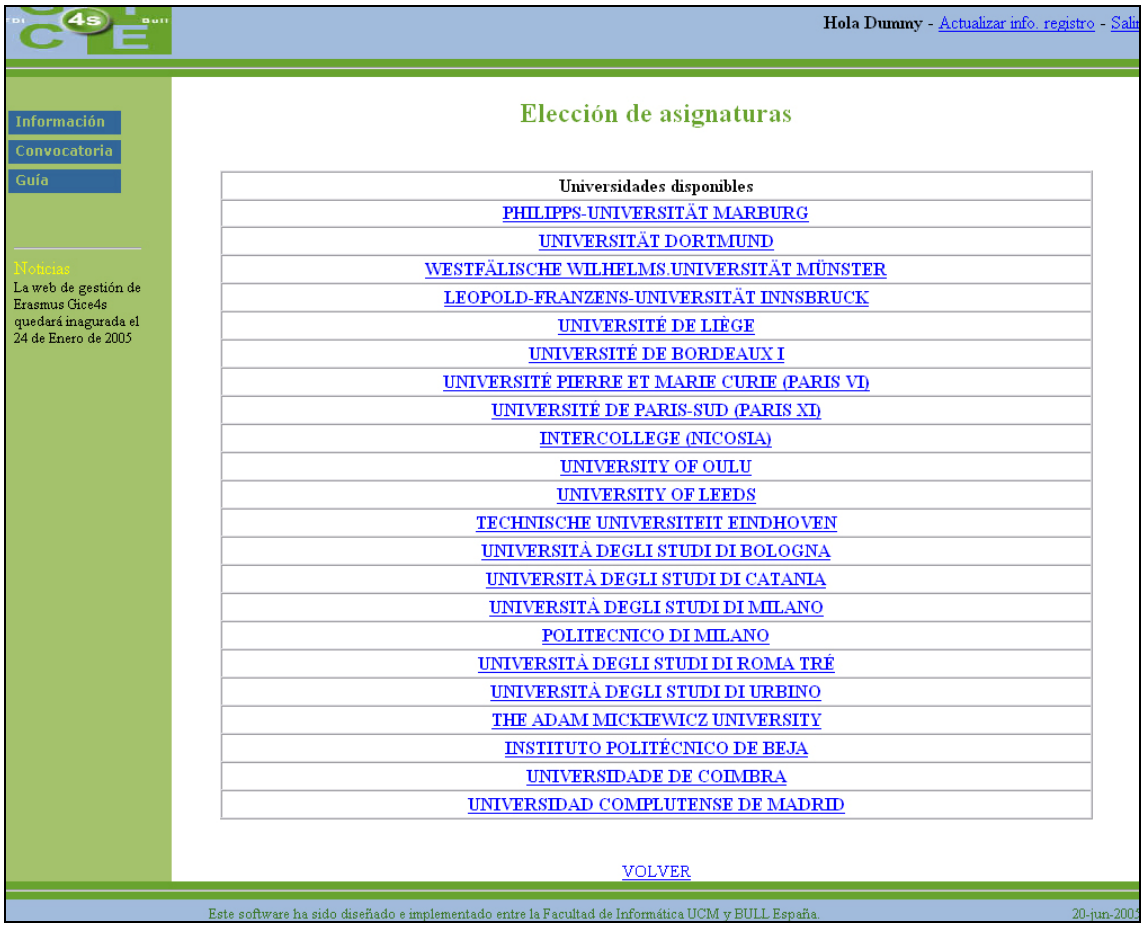


Figura 22: Captura de pantalla de la lista de universidades

Una vez elegida la Universidad, se le mostrará la lista de facultades pertenecientes a la misma que se encuentran en la base de datos. En esta pantalla tiene la opción de realizar las convalidaciones sin usar el asistente o, el caso que nos interesa, usándolo. Además, desde esta pantalla puede acceder a otra que le permitirá añadir a la base de datos nuevas asignaturas con sus descriptores, que el alumno ha encontrado por su cuenta y que no se encuentran disponibles en el sistema. Estas asignaturas deberán ser revisadas y no son consideradas por el asistente hasta que no se produzca este hecho. Ésta es una forma dinámica de aumentar el conocimiento del sistema sin un gran esfuerzo por parte de la Facultad con la pequeña ayuda de los estudiantes Erasmus.

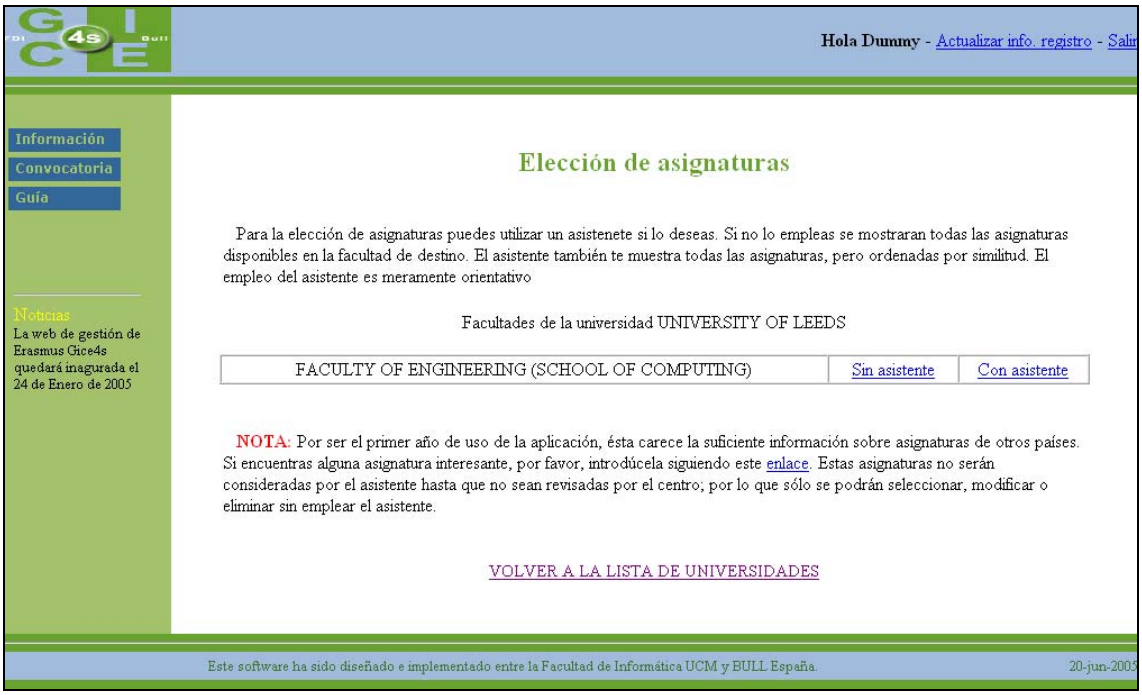


Figura 23: Captura de pantalla de la lista de facultades

4.1.2. Implementación de la elección de la Facultad

Los pasos que sigue la aplicación para realizar la esta tarea están recogidos en el archivo "struts-config.xml", donde aparece la secuencia de acciones y las clases implicadas en cada una de ellas.

Vamos a seguir los pasos que se indican en el archivo de configuración nombrado anteriormente para ver el funcionamiento de esta fase.

1. Clase → `FindAllUniversitiesAction.java`

Esta clase recoge de la base de datos la lista de las universidades que se ofertan como destino. Estos valores son pasados al JSP `MostrarUniversities` para ser mostrados por pantalla.

2. JSP → `MostrarUniversities.jsp`

Este JSP se corresponde con la implementación de la interfaz gráfica de la figura 22, en la que se muestra una lista con todas las universidades de la base de datos para el alumno seleccione la que desee. Pulsando sobre ellas se accede a cada una de las facultades que tienen asociadas.

3. Clase → `FindAllFacultiesAction.java`

Esta clase recoge de la base de datos las facultades asociadas a la universidad que se ha seleccionado en la fase anterior, la cual ha sido pasada como parámetro. Los valores aquí recogidos son pasados al siguiente JSP para ser mostrados por pantalla.

4. JSP → `MostrarFaculties.jsp`

Este JSP implementa la interfaz gráfica de la figura 23, en la que se muestra la lista de las facultades de la universidad seleccionada y se le da al alumno la posibilidad de realizar las convalidaciones con o sin asistente. Desde aquí el enlace que nos interesa en este sistema es el que nos lleva a la acción de sugerir una convalidación , que se explica en la siguiente fase, con sus clases implicadas detalladas.

En esta primera etapa le hemos preguntado al usuario por la universidad y facultad que desea considerar, datos que serán guardados en la sesión para su posterior utilización.

4.2. Elección de la asignatura

4.2.1. Desarrollo de la interfaz

Una vez el alumno ya ha seleccionado una facultad y ha decidido utilizar el asistente, el siguiente paso es elegir qué asignatura de su carrera desea convalidar en su estancia en el extranjero. Para ello se le mostrará una lista de *Radio Buttons* con las asignaturas de su carrera de los últimos cursos.

El alumno seleccionará una de las asignaturas que desee convalidar y le dará al botón “Sugerir asignatura”, pasando así a la siguiente etapa. Esto quiere decir que deberá realizar este proceso para cada una de las asignaturas que desee convalidar, mostrando el sistema unas alternativas distintas para cada una de ellas. Esta forma de seleccionar las asignaturas puede resultar más pesada para el alumno pero, sin embargo, resulta mucho más clara a la hora de mostrar los resultados. Por cada asignatura se muestran ordenadas por similitud todas las asignaturas de la facultad de destino como se comentará en la siguiente etapa. Esto supone que estamos hablando de una cantidad elevada de información que podría resultar confusa si se mostrara para todas las asignaturas a la vez. Este es el motivo por el que se ha optado por utilizar esta forma de seleccionar las asignaturas de la facultad de origen.

Figura 24: Captura de pantalla de la lista de asignaturas

4.2.2. Implementación de la elección de la asignatura

Se mostrará, de igual modo que se hizo para la fase anterior, la secuencia de clases y JSP que intervienen en ésta en el orden de actuación para seguir con mayor claridad la ejecución del sistema.

1. Clase → `MostrarAsignaturasAction.java`

Esta es la primera clase utilizada en el proceso de sugerir convalidaciones por parte del sistema.

Dicha clase es utilizada en diversos puntos del sistema, debido a que recupera las asignaturas que se ofertan en la universidad de destino y las impartidas en la universidad de origen, en este caso la Facultad de Informática. En nuestro caso sólo nos interesa la extracción que realiza de la base de datos relativa a las asignaturas pertenecientes a la universidad de origen, ya que el alumno debe seleccionar de entre las que se mostrarán, aquella que desea que se le convalide una vez finalizada la beca.

2. JSP → `AsistenteAsignaturas.jsp`

Este JSP implementa la interfaz gráfica de la figura 24, en la que se muestra una lista de *Radio Buttons* con las asignaturas de la facultad de origen del alumno, que se han obtenido tras la ejecución de la clase anterior, para que seleccione una de ellas. Una vez seleccionada el alumno pulsará sobre el botón “Sugerir asignatura” que provocará la llamada de la siguiente clase.

3. Clase → `MostrarSugerenciasAction.java`

Esta clase es la que lleva todo el peso de la inteligencia de este sistema utilizando otras auxiliares como `SimilitudAsignaturas.java`, que se explica en detalle en el apartado B de este mismo capítulo.

La primera acción que realiza esta clase es extraer de la base de datos las asignaturas de la universidad y facultad seleccionadas, dato este que se recupera de la sesión ya que se había guardado en la primera etapa de este sistema.

Una vez obtenida la lista de asignaturas de la facultad de destino, se aplica la función de similitud entre cada una de ellas con la asignatura de la facultad de origen. Esto provoca que cada asignatura tenga asociado un valor de similitud. Además, en este proceso se aprovecha para comprobar si la asignatura ya ha sido seleccionada para otra convalidación, marcándose en consecuencia.

La lista resultante se divide en dos sublistas: una con las asignaturas que superan un umbral de similitud y otra con las que no lo hacen. Este umbral es distinto dependiendo del carácter de la asignatura de la facultad de origen, siendo mayor para las obligatorias y troncales que para las optativas. Las asignaturas que superan este umbral se considerarán convalidables.

Las asignaturas que no superan el umbral se guardan en otra lista para hacer una segunda comprobación que hace que el sistema sea más sofisticado. Se comprueban las posibles parejas de asignaturas por si uniendo algunas de ellas se consiguiera una similitud suficiente; lo que querría decir que

serían convalidables si se cursan juntas. Estas parejas de asignaturas se guardan en otra lista.

Esta clase devuelve las tres listas generadas en ella, la lista de las asignaturas que son convalidables por sí mismas, las que serían convalidables si se cursan juntas y las que no son convalidables.

4.3. Elección de la convalidación

4.3.1. Desarrollo de la interfaz

En esta etapa hay que presentarle bastante información al alumno y debe ser de la forma más clara posible. Por este motivo se ha optado por utilizar un código de colores sencillo y aclarativo. Se mostrará una lista de todas las asignaturas de la facultad de destino junto con una puntuación (entre 0 y 100) que corresponderá con la similitud que tiene esa asignatura con la de la facultad de origen que seleccionó el alumno en la etapa anterior.

Como es posible la convalidación de dos asignaturas de otra facultad con una de le de origen también se muestran las combinaciones de dos asignaturas que si se cursan juntas produzcan una similitud elevada con la asignatura que eligió el alumno. En este caso la puntuación mostrada debe ser considerada cuando se seleccionen juntas para la convalidación y no por separado, que tendrán una puntuación inferior.

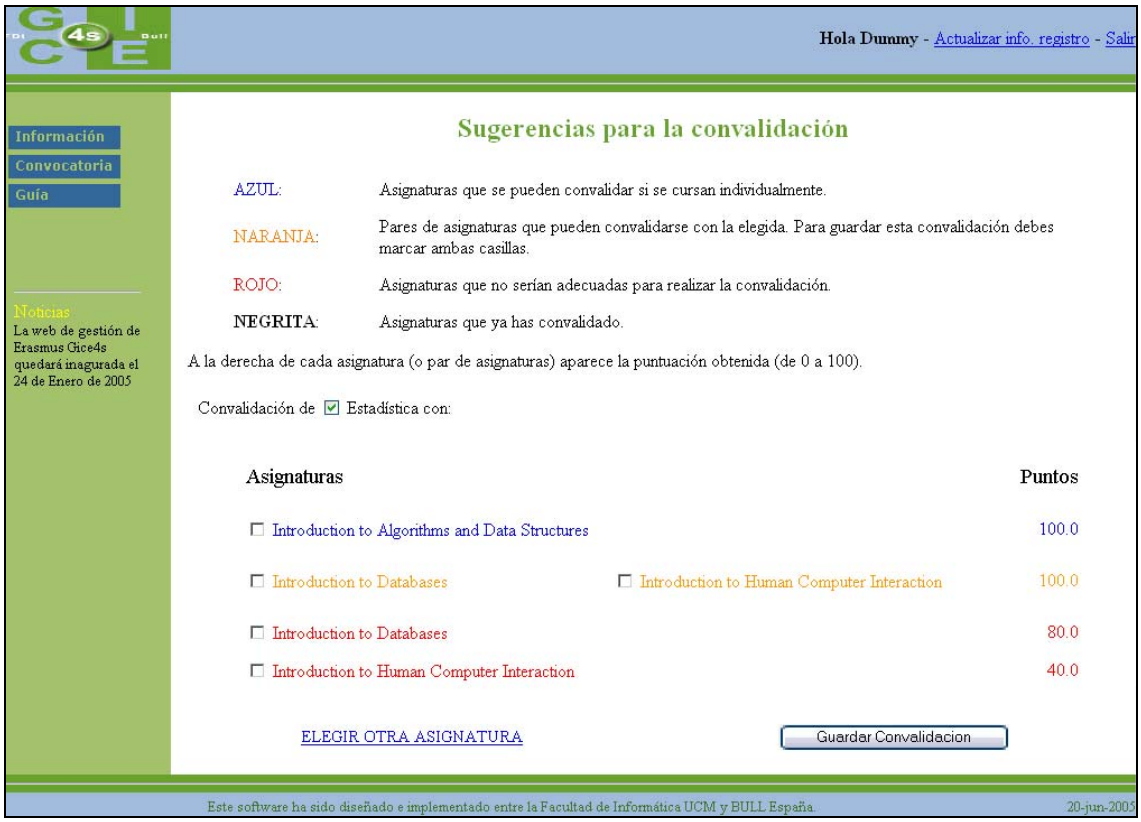


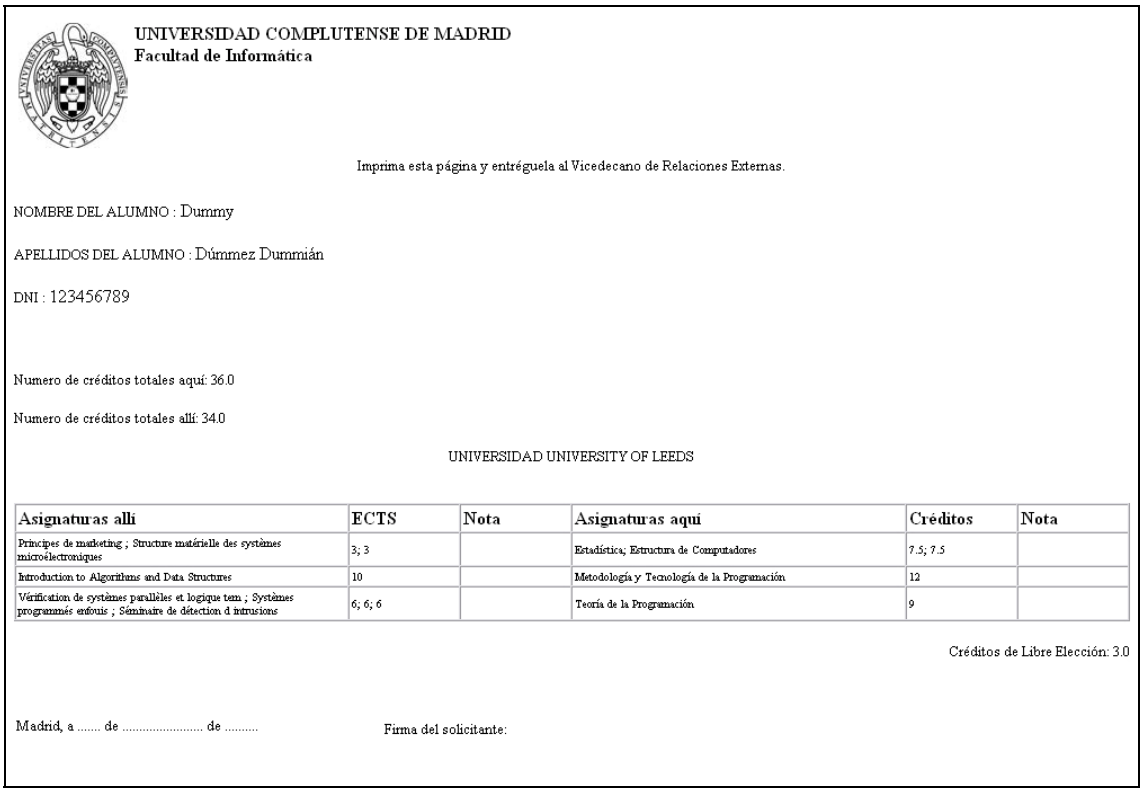
Figura 25: Captura de pantalla de los resultados devueltos por el sistema

El código de colores utilizado es el siguiente:

- **AZUL**: Asignaturas que se pueden convalidar si se cursan individualmente.
- **NARANJA**: Pares de asignaturas que pueden convalidarse con la elegida. Para que se puedan convalidar se deben marcar ambas asignaturas, no una sola.
- **ROJO**: Asignaturas que no serían adecuadas para realizar la convalidación.

Además, se le ofrece al alumno otra ayuda interesante. Se le muestran en **NEGRITA** las asignaturas que ya ha decidido convalidar por otra. De esta forma si ya ha seleccionado en una convalidación anterior una asignatura que le sirve para convalidar la actual el alumno ya lo sabrá. Así se cubre el caso en el que cursando una asignatura en la facultad de destino se puede convalidar por más de una en la de origen.

Una vez mostrada toda esta información, el alumno ahora deberá decidir que asignatura (o asignaturas) escoge para realizar la convalidación marcando el *Check Box* correspondiente y, posteriormente, dando al botón "Guardar Convalidación". Esto provocará que se añada a la lista de convalidaciones que lleva realizada el alumno y que se le mostrará por pantalla en el formato oficial.



UNIVERSIDAD COMPLUTENSE DE MADRID
Facultad de Informática

Imprima esta página y entréguela al Vicedecano de Relaciones Externas.

NOMBRE DEL ALUMNO : Dummy

APELLIDOS DEL ALUMNO : Dúmmez Dummián

DNI : 123456789

Numero de créditos totales aquí: 36.0

Numero de créditos totales allí: 34.0

UNIVERSIDAD UNIVERSITY OF LEEDS

Asignaturas allí	ECTS	Nota	Asignaturas aquí	Créditos	Nota
Principes de marketing ; Structure matérielle des systèmes microélectroniques	3; 3		Estadística; Estructura de Computadores	7.5; 7.5	
Introduction to Algorithms and Data Structures	10		Metodología y Tecnología de la Programación	12	
Vérification de systèmes parallèles et logique tem ; Systèmes programmés enfouis ; Séminaire de détection d intrusions	6; 6; 6		Teoría de la Programación	9	

Créditos de Libre Elección: 3.0

Madrid, a ..... de ..... de .....

Firma del solicitante:

Figura 26: Captura de pantalla de la lista de convalidaciones

4.3.2. Implementación de la elección de la convalidación

En esta etapa se tienen que mostrar los datos obtenidos de la anterior, para ello se muestra la interfaz gráfica de la figura XX:

1. JSP → `Sugerencias.jsp`

Es el JSP que implementa la interfaz. Para ello utiliza las tres listas generadas por la clase `MostrarSugerenciasAction.java` y las muestra por pantalla con un color distintivo para recibir la información importante de forma directa con solo golpe de vista. Además, al lado de cada asignatura muestra la puntuación obtenida.

En esta pantalla el alumno puede seleccionar una asignatura entre las mostradas para realizar la convalidación, dando posteriormente al botón “Guardar Convalidación”, donde el sistema almacenará la convalidación elegida por el usuario, acción que se realiza a través de la clase siguiente.

2. Clase → `selectedSubjectForm.java`

Formulario empleado para almacenar los valores introducidos desde la interfaz, que contiene simplemente atributos y métodos de acceso.

3. Clase → `SelectedSubjectsActions.java`

Esta clase recoge los valores del formulario citado anteriormente, teniendo así los valores de las asignaturas que el alumno quiere ha seleccionado de la universidad de origen y de destino.

Con estos valores el sistema recupera de la base de datos los créditos de las asignaturas citadas, y prepara la convalidación para ser insertada en la misma dentro de una tabla de convalidaciones la cual será revisada posteriormente por el coordinador para comprobar su viabilidad. Esta tabla, *Convalidaciones*, ha sido explicada en el capítulo IV de esta memoria.

4. Clase → `MostrarConvalidacionesAction.java`

Esta clase recupera las convalidaciones de la tabla citada anteriormente, *Convalidaciones*, que corresponden al alumno en cuestión, ya que cada convalidación almacenada está asociada a un usuario en concreto a través del DNI. Estos resultados obtenidos son pasados al siguiente JSP para ser presentados por pantalla.

5. JSP → `MostrarConvalidacionesContent.jsp`

Esta presentación corresponde a la figura 26 mostrada anteriormente. Como puede observarse se muestra la relación de asignaturas que el alumno tiene seleccionadas, así como los créditos de las mismas y el número total de créditos elegidos en los lugares de origen y destino.

Es aquí donde el alumno tiene la posibilidad de rectificar la convalidación en caso de que no sea de su agrado o simplemente quiera corregir errores antes de imprimir el documento, el cual deberá ser entregado al coordinador, tal y como se especifica en las instrucciones presentadas.

5. Validación y verificación

Para comprobar el correcto funcionamiento del sistema se han realizado numerosas pruebas con distintos casos posibles.

Debido a la gran complejidad del sistema, es extremadamente complicado comprobar que el resultado que devuelve es el correcto. Por este motivo se han introducido valores artificiales en la base de datos para tener casos sencillos y manejables, pudiendo controlar así los resultados obtenidos y contrastarlos con los esperados.

Además, se ha probado con la base de datos real para comprobar que no se producía ningún error inesperado provocado por una mayor carga de trabajo.

6. Conclusiones

El sistema de ayuda al alumno para la elección de asignaturas es un sistema muy útil de cara al alumno y que puede hacerle mucho más fácil este proceso en una etapa en la que, muy seguramente, se verá desbordado por la cantidad de trámites y decisiones importantes que deberá tomar para afrontar una experiencia académica y personal que no olvidará nunca.

Para ofrecer esta ayuda primero se pensó en permitir a los alumnos una búsqueda temática de asignaturas de la facultad de destino, como se explica en el capítulo II, pero este sistema es mejor en el sentido de que es más sofisticado y ofrece al alumno directamente lo que está buscando.

En el primer año de uso del sistema no se alcanzarán resultados óptimos debido a la falta de conocimiento sobre las asignaturas de algunas universidades extranjeras y por el conocimiento impreciso de otras. Esto es causado por las limitaciones comentadas sobre el proceso de captura y asignación de descriptores realizado. Esta carencia será fácilmente subsanable en los siguientes años de uso por la aportación de nuevo conocimiento por parte de los alumnos Erasmus que vuelvan de su estancia en el extranjero. Además, se podrá gestionar un histórico con las convalidaciones realizadas en los años venideros para mejorar aún más la fiabilidad y eficiencia del sistema.

E. SISTEMA DE AYUDA AL COORDINADOR ERASMUS PARA DECIDIR CONVALIDACIONES

1. Introducción

En esta funcionalidad también puede apreciarse el módulo de inteligencia artificial del sistema, facilitando, en este caso, la labor al coordinador Erasmus.

Es de gran utilidad ya que supone un gran ahorro de tiempo en el desarrollo de actividades que hasta el momento se realizaban de forma manual.

Actualmente el proceso de definición del acuerdo de reconocimiento académico se realiza artesanalmente.

El estudiante analiza el plan de estudios de la universidad de destino y, de acuerdo a lo que le queda por cursar en sus estudios, selecciona qué asignaturas quiere realizar y por cuáles le gustaría convalidarlas. Presenta esta propuesta al coordinador que es el responsable de darle el visto bueno. Esta propuesta puede sufrir varias iteraciones hasta que sea del agrado de las dos partes, ya que al hacerlo de forma manual hay que tener en cuenta numerosos criterios que en ocasiones se pasan por alto.

A grandes rasgos, el sistema va guiando al coordinador hasta la elección de la universidad de destino de la cual quiere comprobar la viabilidad de una convalidación de asignaturas. Una vez elegidas éstas, mostrará por pantalla los resultados obtenidos en la convalidación e indicará si es posible o no su realización.

2. Especificación de requisitos

Al igual que ocurría con el caso del alumno, es preciso que el coordinador previamente se haya dado de alta en el sistema. Este es un requisito indispensable, ya que si no se encuentra registrado no tiene acceso al mismo.

En cuanto a requisitos del sistema, éste se apoya en una base de datos que ha sido confeccionada atendiendo a las necesidades del mismo.

Para que el sistema funcione correctamente dicha base de datos debe contener una relación de las universidades de destino y origen, y las asignaturas que en ellas se imparten, ya que sin esta información el sistema quedaría inservible.

Algo importante a la hora de decidir que asignaturas son o no convalidables es tener en cuenta los contenidos de las mismas, que en la base de datos se reflejan en forma de descriptores. Es por ello realmente necesario, que éstos se encuentren almacenados en la base de datos y se establezcan asociaciones con sus respectivas asignaturas.

3. Conocimiento utilizado

El conocimiento empleado para esta parte del desarrollo se apoya, entre otras cosas, en la función que establece la similitud entre asignaturas, explicada en el apartado B de este mismo capítulo.

También mencionar el uso de la ontología sobre descriptores del área de la informática, tratada en el capitulo IV y en el apéndice A, pudiéndose consultar estos puntos para más información.

4. Implementación

Para la ejecución de esta funcionalidad el sistema guía al coordinador a través de una serie de interfaces, las cuales se elaboran de manera que los pasos a realizar en cada una de ellas queden lo más claros y sencillos posibles.

Al igual que en el sistema de ayuda para la elección de una universidad de destino, se seguirán la secuencia de operaciones que queda reflejada en el archivo de configuración, aclarando así los pasos que sigue el sistema.

El orden de exposición queda dirigido por el orden de aparición de las distintas interfaces.

4.1. Primera interfaz: Elección de universidad

En esta interfaz se muestra la lista de las universidades que se ofrecen como destino, para que el coordinador seleccione aquella sobre la cual desea comprobar una convalidación con respecto a asignaturas españolas

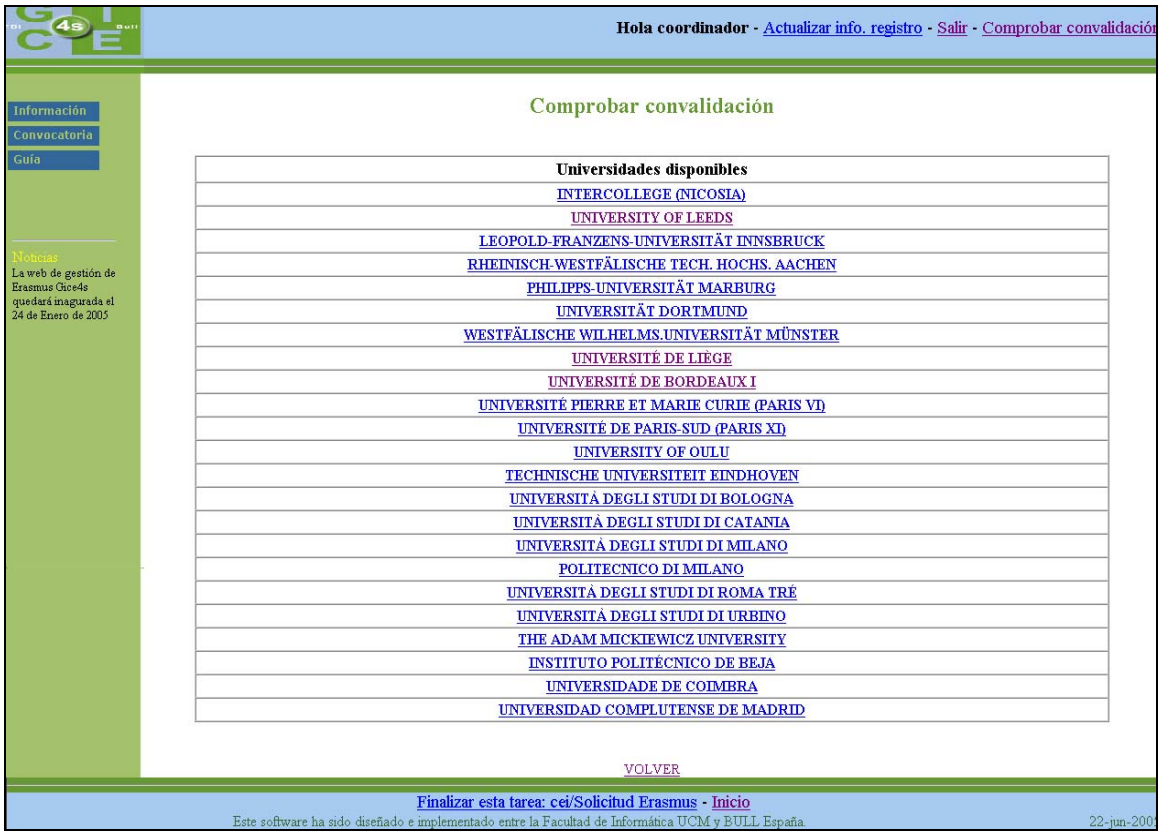


Figura 27: Captura de pantalla del asistente para comprobar convalidación

4.1.1. Implementación

De acuerdo al orden de aparición en el archivo de configuración mencionado las clases, formularios y JSPs implicados serían:

1. Clase → `FindAllUniversitiesAction.java`

Esta clase realiza una consulta a la base de datos para recuperar de ella la lista de las universidades ofertadas. Estos datos son pasados a un JSP (`MostrarUniversities.jsp`) para ser mostrados por pantalla.

2. JSP → `MostrarUniversities.jsp`

El resultado de este JSP se corresponde con la Figura1, mostrada anteriormente.

Como puede observarse se presentan las universidades recogidas mediante la clase anterior. El coordinador selecciona una de las mostradas, pulsando sobre el link que queda sobre cada una de ellas.

El código desglosado puede verse al final del documento.

4.2. Segunda interfaz: Elección de facultad

Figura 28: Captura de pantalla del asistente para comprobar convalidación

Esta interfaz muestra las facultades que pertenecen a la universidad elegida en la etapa anterior.

Actualmente para esta primera versión del proyecto GICE4S, solo se dispone de una facultad por universidad, pero la interfaz está pensada de manera que puedan añadirse sucesivas facultades cuando se encuentre necesario.

4.2.1. Implementación

De acuerdo al orden de aparición en el archivo de configuración mencionado las clases, formularios y JSPs implicados serían:

1. Clase → `FindAllFacultiesAction.java`

Esta clase realiza una consulta a la base de datos para recuperar de ella la lista de las facultades que están asociadas a la universidad seleccionada en la interfaz anterior. Estos datos son pasados a un JSP (`MostrarFaculties.jsp`) para ser mostrados por pantalla.

2. JSP → `MostrarFaculties.jsp`

El resultado de este JSP se corresponde con la Figura2, mostrada anteriormente.

Como puede observarse se presenta una única facultad, ya que por el momento, en esta versión de GICE4S, todas las asignaturas quedan recogidas en ella.

El diseño del JSP ha sido pensado para poder incluir más facultades en la base de datos de manera sencilla, si se desea.

Pulsando sobre el link que queda sobre la facultad se accede a la siguiente interfaz.

El código desglosado puede verse al final del documento.

4.3. Tercera interfaz: Elección de asignaturas para comprobar su convalidación

Figura 29: Captura de pantalla del asistente para comprobar convalidación

Como puede observarse se muestran por pantalla las asignaturas ofertadas en las universidades de origen y destino.

En las asignaturas de la universidad extranjera se ofrece la posibilidad de consultar el temario de la mismas pulsando sobre ellas, completando así la información que se le ofrece al coordinador.

En esta etapa el se deben marcar aquellas combinaciones de asignaturas de las cuales desee comprobar su convalidación, pudiendo seleccionar de n a m, es decir distinto número de asignaturas en el origen y en el destino.

4.3.1. Implementación

De acuerdo al orden de aparición en el archivo de configuración mencionado las clases, formularios y JSPs implicados serían:

1. Clase → `MostrarAsignaturasAction.java`

Esta clase es la encargada de realizar las consultas a la base de datos, para recuperar de ella las asignaturas ofertadas en la universidad de destino, y las impartidas en la universidad de origen, en este caso la Facultad de Informática.

En el caso de las asignaturas de destino, se recupera también el enlace a su programa o temario, de manera que sea sencillo comprobar los contenidos que en ellas se imparten.

2. JSP → `MostrarSubjects.jsp`

El resultado de este JSP se corresponde con la Figura3, mostrada anteriormente.

En esta interfaz el coordinador deberá marcar aquellas asignaturas de las cuales desee comprobar una convalidación, pudiendo seleccionar un número distinto en el origen y en el destino.

Una vez realizada la selección se recogen los valores en formulario `SelectedSubjectForm`.

El código desglosado puede verse al final del documento.

4.4. Cuarta interfaz: Presentación de resultados

En esta fase se muestran los resultados de la convalidación, presentando al coordinador la relación de asignaturas seleccionadas en la etapa anterior y el grado de similitud que presentan atendiendo a los créditos y al contenido de las mismas.

La similitud entre asignaturas puede tomar valores entre 0 y 1. Para determinar si se puede o no realizar una convalidación se toma como umbral un valor de 0.8, ya que debe ser un limite que se acerque al valor máximo, en este caso 1.

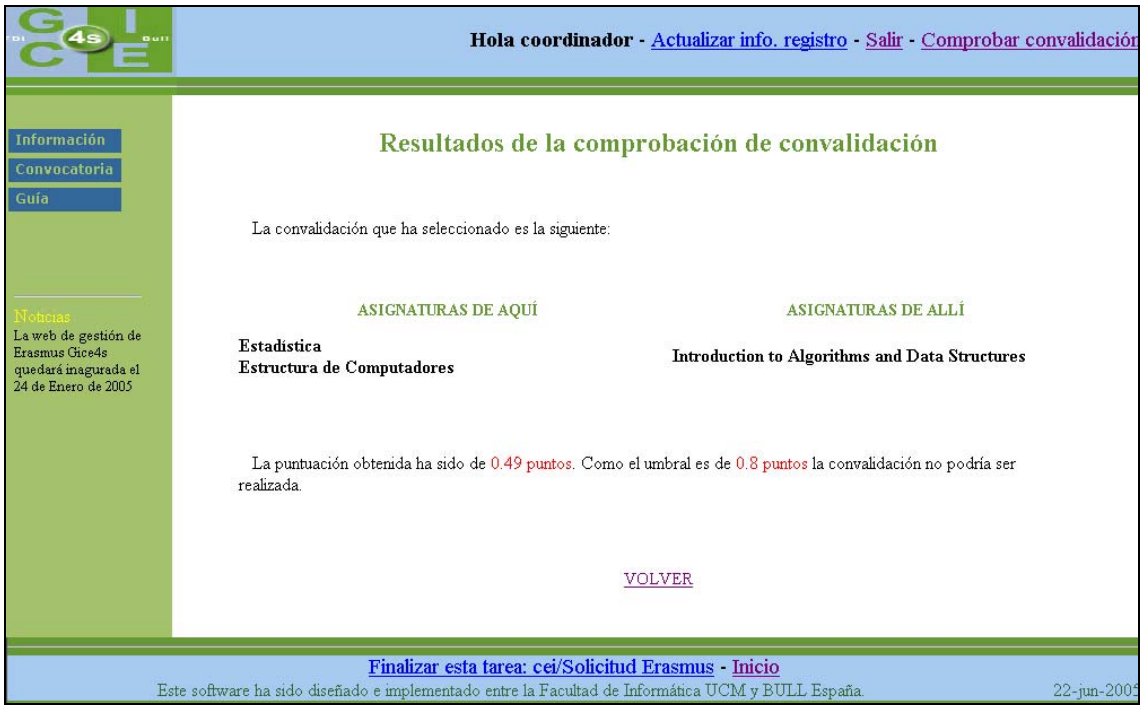


Figura 30: Pantalla de resultados del asistente para comprobar si una asignatura es convalidable

4.4.1. Implementación

De acuerdo al orden de aparición en el archivo de configuración mencionado las clases, formularios y JSPs implicados serían:

1. Clase → `MostrarResultadosConvalidacionAction.java`

Esta es la clase más importante de esta parte del desarrollo, ya que es aquí donde el sistema emplea el módulo de inteligencia artificial.

Primeramente se recogen los valores del formulario `SelectedSubjectForm`, que corresponderán a las asignaturas que serán objeto de estudio.

Para el cálculo de convalidaciones se hace uso de la clase `SimilitudAsignaturas` explicada en el apartado b del capítulo V.

Esta clase dispone de una función que calcula la similitud entre un conjunto de asignaturas, devolviendo un número entre 0 y 1, siendo éstos los valores mínimo y máximo.

Dicha función de similitud debe recibir como parámetros las dos listas de asignaturas que se desean comparar. Para ello hay que hacer uso de otra clase, `Asignatura`, que nos permite convertir los valores recogidos del formulario en objetos de esta clase.

Cada una de las asignaturas tiene como atributos un nombre, un número de créditos y los descriptores que representan los contenidos tratados en la misma. El nombre se consigue directamente del formulario, y los dos últimos parámetros mediante consultas a la base de datos.

Una vez construidos estos objetos, podemos calcular la similitud, que finalmente será comparada con un umbral mínimo para establecer si puede llevarse a cabo la convalidación, siempre y cuando el valor de la similitud supere dicho umbral.

El valor del umbral se ha establecido en 0.8, ya que tras diversas pruebas se ha comprobado que es el adecuado para establecer una convalidación. No obstante este valor puede ser cambiado si se considera oportuno.

Todos estos resultados son pasados al JSP, `ResultadosConvalidacion`, para ser mostrados por pantalla.

2. Formulario → `SelectedSubjectForm.java`

Formulario empleado para almacenar los valores introducidos desde la interfaz, que contiene simplemente atributos y métodos de acceso.

3. JSP → `ResultadosConvalidacion.jsp`

La presentación de resultados del proceso del cálculo de convalidaciones se corresponden con la mostrada en la Figura4.

Como puede observarse se muestran las asignaturas que han sido seleccionadas en la interfaz anterior, junto con la similitud que han obtenido entre ellas.

A modo de aclaración, si la similitud supera o es igual al umbral establecido se presentan los resultados del cálculo de similitudes en color verde, mientras que en el caso contrario se presentarían en color rojo.

Junto con la similitud obtenida, se muestra el umbral empleado.

El código desglosado puede verse al final del documento.

5. Ejemplo del funcionamiento del sistema

A modo de aclaración se presenta un ejemplo del funcionamiento del sistema.

Los pasos a seguir vienen detallados de modo gráfico en las figuras que se van presentando a lo largo de esta parte del desarrollo, de modo que cuando se hace referencias a ellas es porque corresponden exactamente al ejemplo que se está explicando.

Supongamos que el coordinador quiere comprobar la convalidación entre las asignaturas “Estadística” y “Estructura de Computadores” impartidas en la Facultad de Informática, y la asignatura “Introduction to Algorithms and Data Structures” que se ofrece en la universidad de Leeds.

Ante esta situación, el coordinador entraría en el sistema y seleccionaría la universidad de Leeds, de entre las que se muestran en la figura 27.

A continuación se le presenta, tal y como muestra la figura 28, la facultad que está contenida en la universidad de Leeds, y sobre la cual debe pulsar para acceder a las asignaturas.

Se tiene entonces una lista de las asignaturas de ambas universidades, donde en nuestro caso, y tal y como muestra la figura 29, se deben seleccionar las asignaturas de “Estadística” y “Estructura de Computadores” como “Asignaturas Aquí” , y “Introduction to Algorithms and Data Structures” en la parte de “Asignaturas Allí”. A continuación se pulsa sobre el botón “Comprobar convalidación”.

Es entonces cuando se presentan los resultados de la figura 30, donde puede verse como se muestran las asignaturas que hemos elegido y el valor de la similitud entre ellas.

En este caso se obtiene un valor de 0.49, por lo que no supera el umbral de 0.8, con lo que la convalidación no puede ser realizada. Puede observarse como se presentan estos valore en color rojo.

6. Validación y Verificación

Para la validación y verificación se han realizado numerosas pruebas y se ha llegado a la conclusión de que el sistema tiene un comportamiento adecuado.

Las primeras pruebas se realizaron empleando una base de datos reducida en la cual se cargaban datos ficticios introducidos por nosotros, de tal manera que se conocía el resultado esperado. Se introdujeron numerosas combinaciones posibles hasta que se pudo concluir que realmente el sistema funcionaba como debía.

El motivo por el cual se realizaron pruebas pequeñas y con datos preparados es debido al tamaño de la base de datos real, ya que contiene numerosas asignaturas tanto de la universidad de origen como de las de destino, de forma que sería muy complicado comprobar todos los cálculos de similitudes entre las posibles combinaciones.

La complejidad del sistema crece a medida que se introducen nuevas asignaturas en las universidades.

Para determinar el umbral a partir del cual se puede considerar que dos asignaturas son convalidables, se realizaron pruebas comprobando los valores que se obtenían entre distintos conjuntos de asignaturas.

Se eliminaron de las pruebas los grupos de asignaturas cuya semejanza es total, devolviendo un valor de similitud de 1, y los casos en los que las asignaturas a comparar son absolutamente dispares, donde el valor que se obtiene es 0.

Con el rango de combinaciones que quedan, se realizaron uniones haciendo que los conjuntos difirieran en algunos descriptores y en el número de créditos. Al realizar estas similitudes se observó que pasado el valor 0.8 podía considerarse que las asignaturas eran suficientemente iguales en cuanto a créditos y contenidos para poder ser convalidadas.

7. Conclusiones

Una vez comprobado el funcionamiento del sistema puede concluirse que se comporta de manera adecuada proporcionando gran utilidad al coordinador Erasmus.

En la introducción de este mismo punto, ya se mencionó las ventajas que presenta el disponer de un sistema automático para el desarrollo de esta labor, en contraposición con la elaboración manual existente hasta el momento.

La realización de la actividad de convalidación haciendo uso del sistema supone un gran ahorro de tiempo y simplicidad, ya que bastan unos simples minutos para su desarrollo. Un ejemplo de ello se presenta en el punto 5.

En esta primera versión del proyecto GICE4S, se cuenta con una base de datos que carece de la suficiente información como para que el sistema elabore resultados realmente correctos, debido a que no existen asignaturas para todas las universidades, y los descriptores de las mismas no son del todo fiables ya que, como se comento en el apartado de “recuperación de información”, tenemos el problema del idioma a la hora de extraerlos de sus respectivos programas.

Esta es una razón por la cual actualmente el sistema no es del todo fiable, no por su comportamiento interno, sino por el hecho de que se

necesitaría una base de datos más completa en los aspectos que se han mencionado.

El sistema muestra la similitud entre las asignaturas y aconseja que no se convaliden si se supera un cierto umbral. Dicho umbral ha sido determinado tras una serie de pruebas, pero es meramente informativo, dándole la palabra final al coordinador, en cuanto a convalidaciones se refiere.

CAPÍTULO VI:

CONCLUSIONES

CONCLUSIONES

El objetivo del proyecto ha sido crear un módulo d einteligencia artificial para el sistema GICE4S que ayude tanto a alumnos como al coordinador Erasmus en distintas tareas. Para ello se han implementado tres asistentes: uno para ayudar al alumno a elegir universidad de acuerdo con sus preferencias y sus necesidades académicas; otro que lo ayude a elegir asignaturas; y, por último, un asistente que ayuda al coordinador Erasmus a decidir si una asignatura es convalidable o no.

Estos asistentes utilizan una función de similitud entre asignaturas, que ha sido desarrollado anteriormente a partir del conocimiento del dominio, representado en forma de ontología, y de un conocimiento de las asignaturas, almacenado en la base de datos como conjunto de descriptores.

Además, se ha desarrollado una aplicación basado en la recuperación automática de información de textos, que extrae de los programas de las asignaturas alojados en Internet los descriptores que cumple, almacenando el resultado en la base de datos.

Una vez finalizado el proyecto y visto su funcionamiento dentro del sistema GICE4S, puede afirmarse que representa un módulo de inteligencia artificial de gran utilidad, ya que reune las capacidades de razonamiento que se esperaban del sistema.

El módulo desarrollado es capaz de ofrecer una fuente de razonamiento sobre diversas situaciones que hasta el momento se desarrollaban de forma artesanal, es decir, dependiendo del conocimiento humano para su realización, siendo esto un inconveniente en numerosas ocasiones debido al esfuerzo que esto supone. El proyecto aporta una gran ayuda para el alumno que desea optar a una beca Erasmus ya que permite orientar a éste en su elección, teniendo en cuenta tanto sus necesidades académicas como sus preferencias por el país de destino.

Durante el desarrollo del proyecto surgieron una serie de problemas, entre otros el empleo de distintos idiomas en la descripción de los programas de las asignaturas. Esto ha dificultado la asignación de los descriptores para realizar el cálculo de similitud entre las distintas asignaturas. Para solucionar el problema del idioma sería interesante llegar a un acuerdo para que todos los programas fueran accesibles en inglés, además de incluir una serie de descriptores que cumple la asignatura extraidos de un conjunto estándar, para conseguir que los temarios sean homogéneos. Este problema también puede solventarse con la utilización de EuroWordNet, que puede servir como puente entre distintas lenguas. Este tema se ha explorado, pudiendo ver las conclusiones en el Apéndice B.

Por último cabe destacar la necesidad de disponer de una base de datos con la información suficiente para sacar el máximo rendimiento a la aplicación.

Crear esta base de datos supone una elevada carga de trabajo que queda fuera de los objetivos principales del proyecto.

En definitiva, el módulo de inteligencia artificial del sistema funciona correctamente pero tiene como principal limitación el conocimiento del que dispone. Dado que para que el funcionamiento fuese óptimo necesitaría conocer todas las asignaturas de todas las facultades pertenecientes al programa Erasmus a un nivel muy detallado, no se ha conseguido alcanzar este nivel de eficiencia en esta primera versión. En los sucesivos años se deberá incorporar el conocimiento necesario para que mejore el rendimiento de este módulo. Esta ayuda se puede conseguir de los alumnos que vuelven de su estancia en el extranjero.

Además, en los años sucesivos se podrá incorporar un histórico que guarde las convalidaciones realizadas, así como las distintas sugerencias que ha hecho el sistema a los alumnos en cuanto a universidades y asignaturas, junto con la conformidad de estos con ellas.

BIBLIOGRAFÍA

BIBLIOGRAFÍA

Las direcciones donde se puede encontrar información acerca de las principales tecnologías mencionadas en este documento son las siguientes:

J2EE
- Sun Microsistems (http://java.sun.com/j2ee)

OWL
- Recomendación del W3C (http://www.w3.org/TR/2004/REC-owl-features-20040210)

PROTEGE
- Universidad de Stanford (http://protege.stanford.edu)

DIG
- SourceForge (http://sourceforge.net/projects/dig)

RACER
- Universidad de Concordia (http://www.cs.concordia.ca/~haarslev/racer)
- Racer Systems (http://www.racer-systems.com)

FACT
- Universidad de Manchester (http://www.cs.man.ac.uk/~horrocks/FaCT)

PELLET
- Mindswap (http://www.mindswap.org/2003/pellet)

JENA
- SourceForge (http://jena.sourceforge.net/how-to/dig-reasoner.html)

RICE
- Ronald Cornet (http://www.b1g-systems.com/ronald/rice)
- RICE Manual (Ronald Cornet)

IS
- Universidad de Manchester (http://instancestore.man.ac.uk)

MYSQL
- MySQL (http://www.mysql.com)

JONAS
- ObjectWeb (http://jonas.objectweb.org)
- Evidian y Bull (http://www.evidian.com/jonas)

ECLIPSE
- Eclipse Fpundation (http://www.eclipse.org)

WORDNET

- Universidad de Princeton (http://wordnet.princeton.edu/w3wn.htm)
- Red Iris – Estudio de Lingüística del Español (http://elies.rediris.es/elies2/cap332.htm)

EUROWORDNET

- Universidad de Ámsterdam (http://www.illc.uva.nl/EuroWordNet)
- Red Iris – Estudio de Lingüística del Español (http://elies.rediris.es/elies2/cap334.htm, http://elies.rediris.es/elies8/cap3-2.html)

Las direcciones de las Universidades:

- LEOPOLD-FRANZENS-UNIVERSITÄT INNSBRUCK
  http://www.uibk.ac.at/

- UNIVERSITÉ DE LIÈGE
  http://www.ulg.ac.be/

- INTERCOLLEGE
  http://www.intercollege.ac.cy/

- RHEINISCH-WESTFÄLISCHE TECHNISCHE HOCHSCHULE AACHEN
  http://www.rwth-aachen.de

- UNIVERSITÄT DORTMUND
  http://www.uni-dortmund.de

- PHILIPPS-UNIVERSITÄT MARBURG
  http://www.uni-marburg.de

- WESTFÄLISCHE WILHELMS.UNIVERSITÄT MÜNSTER
  http://www.uni-muenster.de

- UNIVERSITY OF OULU
  http://www.oulu.fi/

- UNIVERSITÉ DE BORDEAUX I
  http://www.u-bordeaux1.fr

- UNIVERSITÉ PIERRE ET MARIE CURIE (PARIS VI)
  http://www.admp.jussieu.fr

- UNIVERSITÉ DE PARIS-SUD (PARIS XI)
  http://www.u-psud.fr

- UNIVERSITY OF LEEDS
  http://www.leeds.ac.uk

- UNIVERSITÀ DEGLI STUDI DI BOLOGNA
  http://www.unibo.it

- UNIVERSITÀ DEGLI STUDI DI CATANIA
  http://www.unict.it

- UNIVERSITÀ DEGLI STUDI DI MILANO
  http://www.unimi.it

- POLITECNICO DI MILANO
  http://www.polimi.it

- UNIVERSITÀ DEGLI STUDI DI ROMA TRÉ
  http://www.uniroma3.it

- UNIVERSITÀ DEGLI STUDI DI URBINO
  http://www.uniurb.it

- TECHNISCHE UNIVERSITEIT EINDHOVEN
  http://www.tue.nl

- THE ADAM MICKIEWICZ UNIVERSITY
  http://www.amu.edu.pl

- INSTITUTO POLITÉCNICO DE BEJA
  http://www.ipbeja.pt

- UNIVERSIDADE DE COIMBRA
  http://www.uc.pt

Direcciones del proyecto GICE4S:

- http://federlin.sip.ucm.es:2001/twiki/bin/view/GICES/WebHome
- http://federlin.sip.ucm.es:2001/twiki/bin/view/GICES/ProcesoGestionReconocimiento
- http://gice4s.fdi.ucm.es/MainPage.do

Otras direcciones de interés:

- Docuemento «Computing Curricula – Computer Engineering» elaborado por la IEEE Computer Society y ACM (http://www.eng.auburn.edu/ece/CCCE/)
- Dirección de consulta de distancias entre dos ciudades (http://www.tutiempo.net/tutiempo.php?pagina=distancias#)

APÉNDICES

A. ONTOLOGÍA

1. INTRODUCCIÓN

Este apéndice está dedicado al desarrollo de la estructura de la ontología implementada en el proyecto para el cálculo de similitud entre asignaturas.

2. ESTRUCTURA DE LA ONTOLOGÍA

La ontología está formada por cinco áreas de conocimiento que forman la Ingeniería Informática. Cada área de conocimiento está constituida a su vez por distintas categorías que incluyen los convenios que deben englobar cada una de ellas. De esta manera, asignando a cada asignatura una serie de convenios, podemos encuadrarla en una o varias áreas de conocimiento y así clasificarla dentro de la ontología.

La ontología parte de una clase raíz (Domain_entity), que contiene dos subclases:

- Hierarchy: Engloba las distintas áreas de conocimientos anteriormente mencionadas, clasificadas en diferentes categorías. Consta de cinco áreas de conocimiento:
    - Hardware
    - Mathematics
    - Software
    - Physics
    - Others
- Topics: Engloba las características que han de cumplir las distintas instancias para ser clasificadas en una u otra área de conocimiento. Están organizadas respetando la jerarquía anterior. Hay un total de 187 topics contenidos en cinco categorías:
    - Hardware_Topics
    - Mathematics_Topics
    - Software_Topics
    - Physics_Topics
    - Others_Topics

A continuación se muestra la estructura completa de la ontología:

- Hierarchy:
  - Hardware: Engloba las distintas categorías relacionadas con el área del hardware.
    - CAO-Computer_Architecture_and_Organization
    - CN-Computer_Networks
    - DL-Digital_Logia
    - ES-Embedded_Systems
    - OS-Operating_Systems
    - VDF-VLSI_Design_and_Fabrication
  - Mathematics: Engloba las distintas categorías relacionadas con el área de las matemáticas.
    - DS-Discrete_Structures
    - PS-Probability_and_Statistics
  - Software: Engloba las distintas categorías relacionadas con el área del software.
    - A-Algorithms
    - CSE-Computer_Systems_Engineering
    - DBS-Database_Systems
    - HCI-Human_Computer_Interaction
    - PF-Programming_Fundamentals
    - SE-Software_Engineering
  - Physics: Engloba las distintas categorías relacionadas con el área de la física.
    - CS-Circuits_and_Signals
    - DSP-Digital_Signal_Processing
    - E-Electronics

- Others: Otras
  - SPI-Social_and_Professional_Issues

- Topics:
  - Hardware_Topics
    - CAO-Computer_Architecture_and_Organization_Topics
      - CAO0-History_and_Overview
      - CAO1-Fundamental_of_Computer_Architecture
      - CAO10-Performance_Enhancements
      - CAO2-Computer_Arithmetic
      - CAO3-Memory_System_Organization_and_Architecture
      - CAO4-Intefacing_and_Communication
      - CAO5-Device_Subsystems
      - CAO6-Processor_Systems_Design
      - CAO7-Organization_of_the_CPU
      - CAO8_Performance
      - CAO9-Distributed_System_Models
    - CN-Computer_Networks_Topics
      - CN0-History_and_Overview
      - CN1-Communications_Network_Architecture
      - CN10-Compression_and_Decompression
      - CN2-Communications_Network_Protocols
      - CN3-Local_and_Wide_Area_Networks
      - CN4-Client_Server_Computing
      - CN5-Data_Security_and_Integrity
      - CN6-Wireless_and_Mobile_Computing

- CN7-Performance_Evaluation
- CN8-Data_Communication
- CN9-Network_Management

- DL-Digital_Logic_Topics
  - DL0-History_and_Overview
  - DL1-Switching_Theory
  - DL10-Design_of_Testability
  - DL2-Combinational_Logic_Circuits
  - DL3-Modular_Design_of_Combinational_Circuits
  - DL4-Memory_Elements
  - DL5-Sequential_Logic_Circuits
  - DL6-Digital_Systems_Design
  - DL7-Modeling_and_Simulation
  - DL8-Formal_Verifications
  - DL9-Fault_Models_and_Testing
- ES-Embedded_Systems_Topics
  - ES0-History_and_Overview
  - ES1-Embedded_Microcontrollers
  - ES10-Interfacing_and_Mixed-signal_Systems
  - ES2-Embedded_Programs
  - ES3-Real-time_Operating_Systems
  - ES4-Low-power_Computing
  - ES5-Reliable_System_Design
  - ES6-Design_Methodologies
  - ES7-Tool_Support

- ES8-Embedded_Multiprocessors
- ES9-Networked_Embedded_Systems

- OS-Operating_Systems_Topics
  - OS0-History_and_Overview
  - OS1-Desing_Principles
  - OS2-Concurrency
  - OS3-Scheduling_and_Dispatch
  - OS5-Device_Management
  - OS6-Security_and_Protection
  - OS7-File_Systems
  - OS8-Systems_Performace_Evaluation
  - OS4-Memory_Management
- VDF-VLSI_Design_and_Fabrication_Topics
  - VDF0-History_and_Overview
  - VDF1-Electronic_Properties_of_Materials
  - VDF10-Semi-custom_Design_Technologies
  - VDF11-ASIC_Design_Methodology
  - VDF2-Function_of_the_Basic_Inverter_Structure
  - VDF3-Combinational_Logic_Structures
  - VDF4-Sequential_Logic_Structures
  - VDF5-Semiconductor_Memories_and_Array_Structures
  - VDF6-Chip_Input-Output_Circuits
  - VDF7-Processing_and_Layout
  - VDF8-Circuit_Characterization_and_Performance
  - VDF9-Alternative_Circuits_Structures-Low_Power_Design

- Mathematics_Topics
  - DS-Discrete_Structures_Topics
    - DS0-History_and_Overview
    - DS1-Functions_Relations_and_Sets
    - DS2-Basic_Logic
    - DS3-Proof_Techniques
    - DS4-Basics_of_Counting
    - DS5-Graphs_and_Trees
    - DS6-Recursion
  - PS-Probability_and_Statistics_Topics
    - PS0-History_and_Overview
    - PS1-Discrete_Probability
    - PS2-Continuous_Probability
    - PS3-Expectation
    - PS4-Stochastic_Processes
    - PS5-Sampling_Distributions
    - PS6-Estimation
    - PS7-Hypothesis_Test
    - PS8-Correlation_and_Regression
- Software_Topics
  - A-Algorithms_Topics
    - A0-History_and_Overview
    - A1-Basic_Algorithmic_Analysis
    - A2-Algorithmic_Strategies
    - A3-Computing_Algorithms

- A4-Distributed_Algorithms
- A5-Algorithmic_Compexity
- A6-Basic_Computability_Theory

o CSE-Computer_Systems_Engineering_Topics

- CSE0-History_and_Overview
- CSE1-Life_Cycle
- CSE10-Specialized_Systems
- CSE11-Reliability_and_Fault_Tolerance
- CSE2-Required_Analysis_and_Elicitation
- CSE3-Specification
- CSE4-Architectural_Design
- CSE5-Testing
- CSE6-Maintenance
- CSE7-Project_Management
- CSE8-Concurrent_Design
- CSE9-Implementation

o DBS-Database_Systems_Topics

- DBS0-History_and_Overview
- DBS1-Database_Systems
- DBS2-Data_Modeling
- DBS3-Relational_Databases
- DBS4-Database_Query_Languages
- DBS5-Relational_Database_Design
- DBS6-Transaction_Processing
- DBS7-Distributed_Database

- DBS8-Physical_Database_Design

- HCI-Human_Computer_Interaction_Topics
  - HCI0-History_and_Overview
  - HCI1-Foundations_of_Human-Computer_Interaction
  - HCI10-Multimedia_Systems
  - HCI2-Graphical_User_Interface
  - HCI3-IO_Tecnologies
  - HCI4-Intelligent_Systems
  - HCI5-Human-centered_Software_Evaluation
  - HCI6-Human-centered_Software_Development
  - HCI7-Interactive_Graphical_User_Interface_Design
  - HCI8-Graphical_User_Interface_Programming
  - HCI9-Graphics_and_Visualizations
- PF-Programming_Fundamentals_Topics
  - PF0-History_and_Overview
  - PF1-Programming_Paradigms
  - PF2-Prgramming_Constructs
  - PF3-Algorithms_and_Problem-solving
  - PF4-Data_Structures
  - PF5-Recursion
  - PF6-Object-oriented_Programming
  - PF7-Event-driven_and_Concurrent_Programming
  - PF8-Using_APIs
- SE-Software_Engineering_Topics
  - SE0-History_and_Overview

- SE1-Software_Processes
- SE2-Software_Requeriment_and_Specifications
- SE3-Software_Design
- SE4-Software_Testing_and_Validation
- SE5-Software_Evolution
- SE6-Software_Tools_and_Environments
- SE7-Languages_Translation
- SE8-Software_Project_Management
- SE9-Software_Fault_Tolerance

- Physics_Topics
  - CS-Circuits_and_Signals_Topics
    - CS0-History_and_Overview
    - CS1-Electrical_Quantities
    - CS2-Resistive_Circuits_and_Networks
    - CS3-Reactive_Circuits_and_Networks
    - CS4-Frequency_Response
    - CS5-Sinusoidal_Analysis
    - CS6-Convolution
    - CS7-Fourier_Analysis
    - CS8-Filters
    - CS9-Laplace_Transforms
  - DSP-Digital_Signal_Processing_Topics
    - DSP0-History_and_Overview
    - DSP1-Theories_and_Concepts
    - DSP10-Audio_Processing

- DSP11-Image_Processing
- DSP2-Digital_Spectra_Analysis
- DSP3-Discrete_Fourier_Transforms
- DSP4-Sampling
- DSP5-Window_Functions
- DSP6-Transforms
- DSP7-Digital_Filters
- DSP8-Discrete_Time_Signals
- DSP9-Convolution

o E-Electronics_Topics

- E0-History_and_Overview
- E1-Electronic_Properties_of_Materials
- E10-Circuit_Modeling_and_Simulation
- E11-Data_Conversion_Circuits
- E12-Electronic_Voltages_and_Current_Sources
- E13-Amplifiers_Design
- E14-Integrated_Circuits_Building_Blocks
- E2-Diodes_and_Diode_Circuits
- E3-MOS_Transistors_and_Biasing
- E4-MOS_Logic_Families
- E5-Bipolar_Transistors_and_Logic_Families
- E6-Design_Parameter_and_Issues
- E7-Storages_Elements
- E8-Interfacing_Logic_Families_and_Standard_Buses
- E9-Operational_Amplifiers

- Others_Topics
  - SPI-Social_and_Professional_Issues_Topics
    - SPI0-History_and_Overview
    - SPI1-Public_Policy
    - SPI2-Methods_and_Tools_of_Analysis
    - SPI3-Professional_and_Ethical_Responsabilities
    - SPI4-Risks_and_Liabilities
    - SPI5-Intellectual_Property
    - SPI6-Privacy_and_Civil_Liberties
    - SPI7-Computer_Crime
    - SPI8-Economic_Issues_in_Computing
    - SPI9-Philosophical_Frameworks

A continuación se muestra una estructura gráfica de la jerarquía anterior:

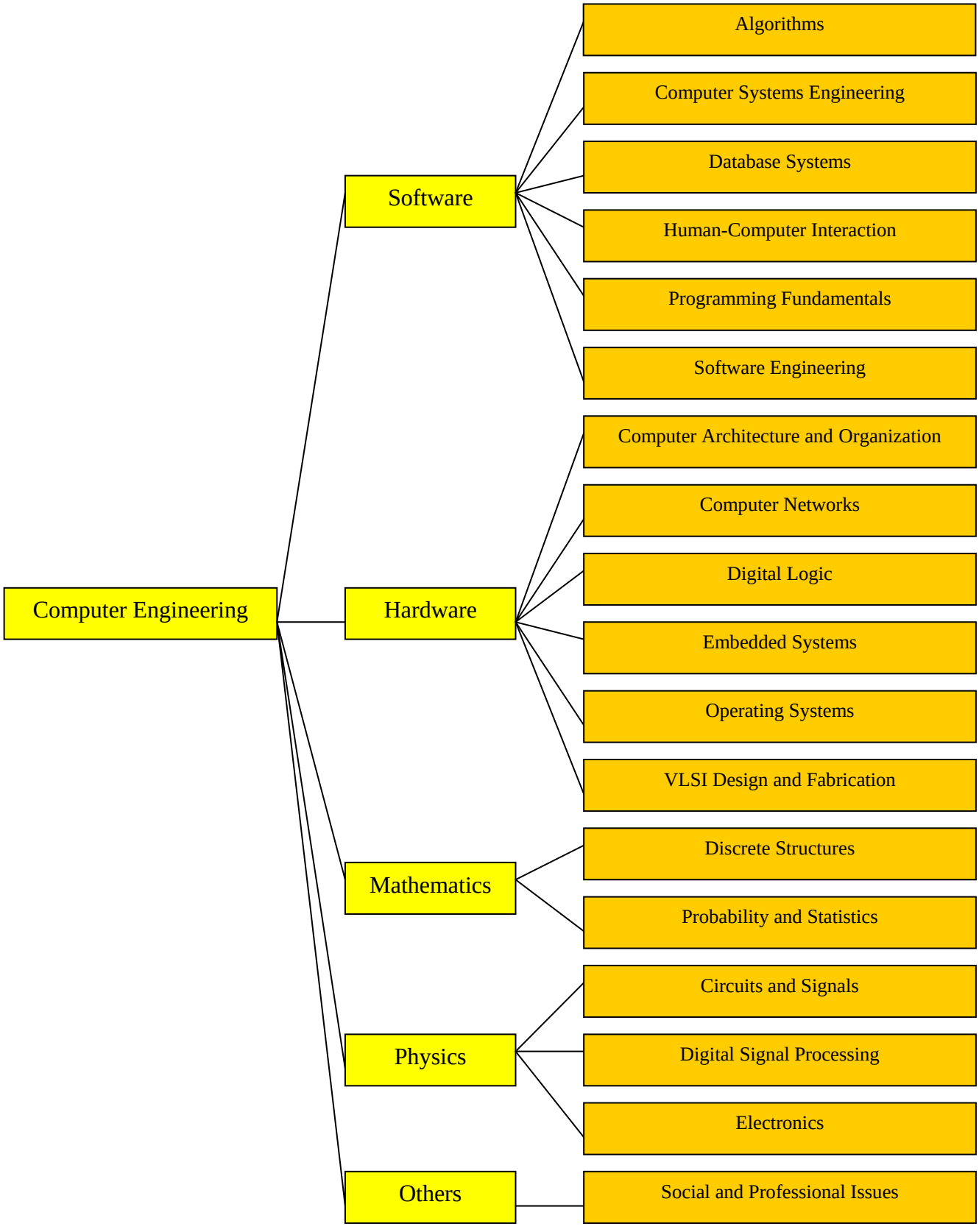


Figura 31: Diagrama que representa la jerarquía de la ontología

Para la definición de propiedades se ha usado la siguiente estructura:

- has_Topic: Propiedad raíz
  - has_Software_Topic: Propiedades del área de Software
    - has_Database_Systems_Topic
    - has_Programming_Fundamentals_Topic
    - has_Human_Computer_Interaction_Topic
    - has_Algorithms_Topic
    - has_Software_Engineering_Topic
    - has_Computer_Systems_Engineering_Topic
  - has_Hardware_Topic: Propiedades del área de Hardware
    - has_Computer_Architecture_and_Organization_Topic
    - has_Digital_Logic_Tepic
    - has_VLSI_Design_and_Fabrication_Topic
    - has_Embedded_Systems_Topic
    - has_Computer_Networks_Topic
    - has_Operating_Systems_Topic
    - has_Physics_Topic: Propiedades del área de Física
  - has_Electronics_Topic
    - has_Circuits_and_Signals_Topic
    - has_Digital_Signal_Processing_Topic
  - has_Others_Topic: Propiedades de otras áreas
    - has_Social_and_Professional_Issues_Topic
  - has_Mathematics_Topic: Propiedades del área de las matemáticas
    - has_Discrete_Structures_Topic
    - has_Probability_and_Statistics_Topic

Cada propiedad parte del dominio de la jerarquía que engloba las distintas áreas de conocimiento (HIERARCHY), y los valores posibles del rango corresponden a cada una de las áreas, dependiendo de la propiedad definida.

DOMINIO (Hierarchy) → RANGO (área de conocimiento)

A parte, a cada una de las áreas de conocimiento se le han asignado restricciones necesarias y suficientes, de modo que para que una instancia (asignatura), sea clasificada dentro de ella ha de cumplir una serie de condiciones. En este caso se ha establecido que para que una asignatura pertenezca a una determinada área basta con que cumpla tres de las condiciones (topics) que en ella se den, de forma que la restricción queda de la siguiente manera:

Has_Nombre del área_Topic >=3

B. EUROWORDNET

1. INTRODUCCIÓN

EuroWordNet se inició en 1996 tomando como pauta a seguir WordNet1.5, incluida dentro de Princeton WordNet. Finalizó su desarrollo en el verano de 1999.

Es un proyecto financiado por la C.E. (LE-2 4003) para el Desarrollo de una Base de Datos Multilingüe con relaciones semánticas entre palabras, pertenecientes a los idiomas inglés, holandés, italiano, español, francés, alemán, checo y estonio. En sus comienzos solo incluía los 4 primeros idiomas.

EuroWordNet propone una mayor riqueza de relaciones semánticas así como diversas soluciones para solventar las deficiencias de WordNet 1.5 tales como la multiplicación innecesaria de sentidos o la falta de consistencia en la aplicación de ciertas relaciones semánticas.

2. WORDNET 1.5

WordNet es una base de datos que contiene una red semántica del inglés. Ha sido desarrollada por George Miller y su grupo de investigación en la Universidad de Princeton (Miller et al. 1993).

Está diseñada utilizando un modelo semántico relacional y una base de datos igualmente relacional y ha sido elaborada desde el punto de vista de la Psicolingüística.

Está considerada como la red semántica del inglés más completa que existe.

Contiene información a cerca de nombres, verbos, adjetivos y adverbios organizados alrededor del concepto de sinónimo (vocablo o de una expresión que tiene una misma o muy parecida significación que otro). Una palabra tendrá asociados un conjunto de sinónimos, y al grupo formados por éstos y la propia palabra es lo que llamamos *synset*.

El vocabulario comprende todas las palabras básicas de cada idioma y los significados y conceptos que son necesarios para relacionar significados más específicos.

A continuación se muestra un ejemplo del funcionamiento de WordNet 1.5

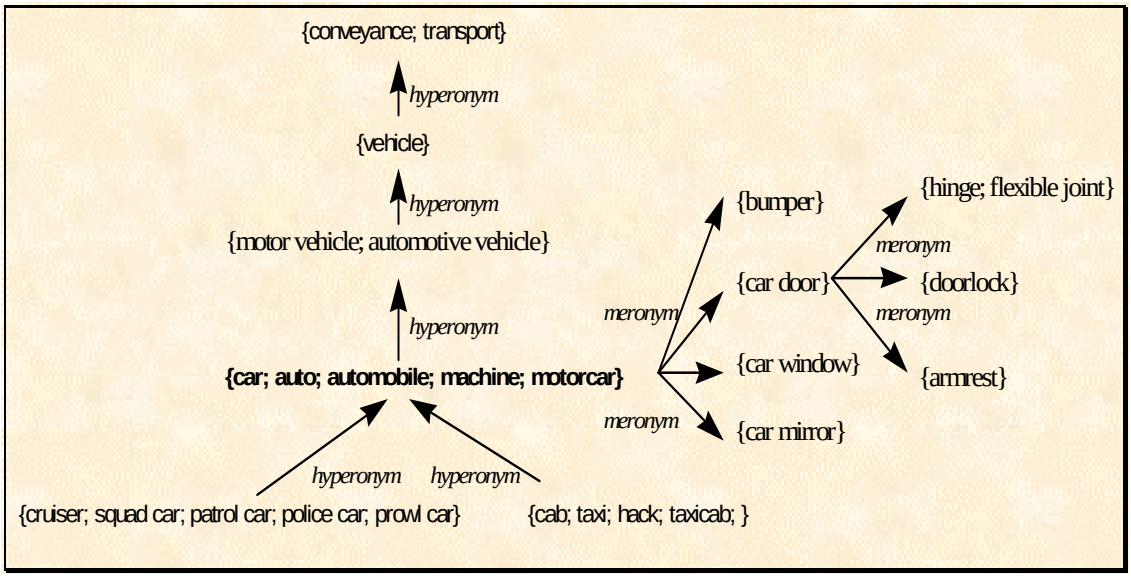


Figura 32: Ejemplo del funcionamiento de WordNet 1.5

Como puede observarse {car; auto; automobile; machine; motorcar} forman un conjunto de sinónimos porque todos ellos pueden referirse al mismo concepto.

Un synset se relaciona con otros mediante relaciones semánticas, tales como hipónimos (relaciones entre conceptos específicos y más generales) o metonimias (relaciones entre partes).

En el ejemplo, el synset {car; auto; automobile; machine; motorcar} esta relacionado con:

- Un synset más general a través de la relación de hiponimia: {motor vehicle; automotive vehicle}
- Un synset más específico: {cruiser; squad car; patrol car; police car; prowl car} y {cab; taxi; hack; taxicab}
- Relaciones de parte: {bumper}; {car door}, {car mirror} and {car window}

3. OBJETIVOS PRINCIPALES DE EWN

Con el desarrollo de la informática y de las telecomunicaciones, la información se está almacenando cada vez más en formatos electrónicos, a los que se accede a través de las redes.

La recuperación de dicha información queda determinada por las palabras clave que se introduzcan en su búsqueda, originando dificultades si no se emplea la terminología adecuada. En el caso de Europa, donde existe diversidad de lenguas, esto supone un inconveniente.

El proyecto EuroWordNet es pues un proyecto de investigación y desarrollo, que tiene como principal objetivo el construir una base de datos multilingüe del vocabulario general, con relaciones semánticas entre palabras de distintas lenguas europeas.

En este sentido presenta tres puntos de partida para comenzar el desarrollo:

- Se trata de crear redes semánticas.
- Dichas redes deben tener un carácter multilingüe que permita ir de una lengua a otra.
- El proyecto no persigue servir para una aplicación específica, sino para todo tipo de aplicaciones dentro de la Recuperación de Información, la Inteligencia Artificial …

4. DISEÑO DE UNA BASE DE DATOS MULTILINGÜE

Para la explicación del funcionamiento de EuroWordNet se adjunta un esquema que será de utilidad para la aclaración de los distintos conceptos.

Cada uno de los idiomas se denomina *wordnet,* y refleja las relaciones semánticas en un sistema interno a la lengua.

Estarán conectados con un índice de significados (en la figura aparece como Inter_Lingual_Index) formado a partir de WordNet 1.5 de inglés americano.

De esta forma cada significado de las distintas palabras estará enlazado con el concepto “más cercano” que aparezca en el índice. Para la determinación del “más cercano” se emplean relaciones semánticas.

Haciendo uso de estas relaciones, es posible ir de una lengua a otra, lo cual hace posible comparar los distintos *wordnets* para descubrir inconsistencias y diferencias entre las distintas lenguas.

Se une también una ontología común independiente del lenguaje y con etiquetas de los campos de conocimiento en los que se usa esa cada uno de los significados. Posee 63 distinciones semánticas.

Esta ontología proporciona un entorno semántico común para todos los idiomas, mientras que las propiedades específicas de cada lengua se mantienen individualmente en los *wordnets*, manteniendo así las diferencias culturales y lingüísticas entre todas ellas.

De esta manera, la base de datos multilingüe se considera un mapa semántico multidimensional, ya que en un *wordnet* individual cada palabra se

coloca con relación al resto de las palabras en esa lengua y, en la base de datos multilingüe, estas redes relativas se conectan entre ellas a través de un índice de significados del inglés.

La principal ventaja de la base de datos es que ofrece la posibilidad de mejorar la llamada de recuperación de información que hace el usuario con una serie de palabras claves, expandiendo dicha consulta a un conjunto más amplio de variantes y palabras relacionadas en cualquiera de las lenguas interconectadas.

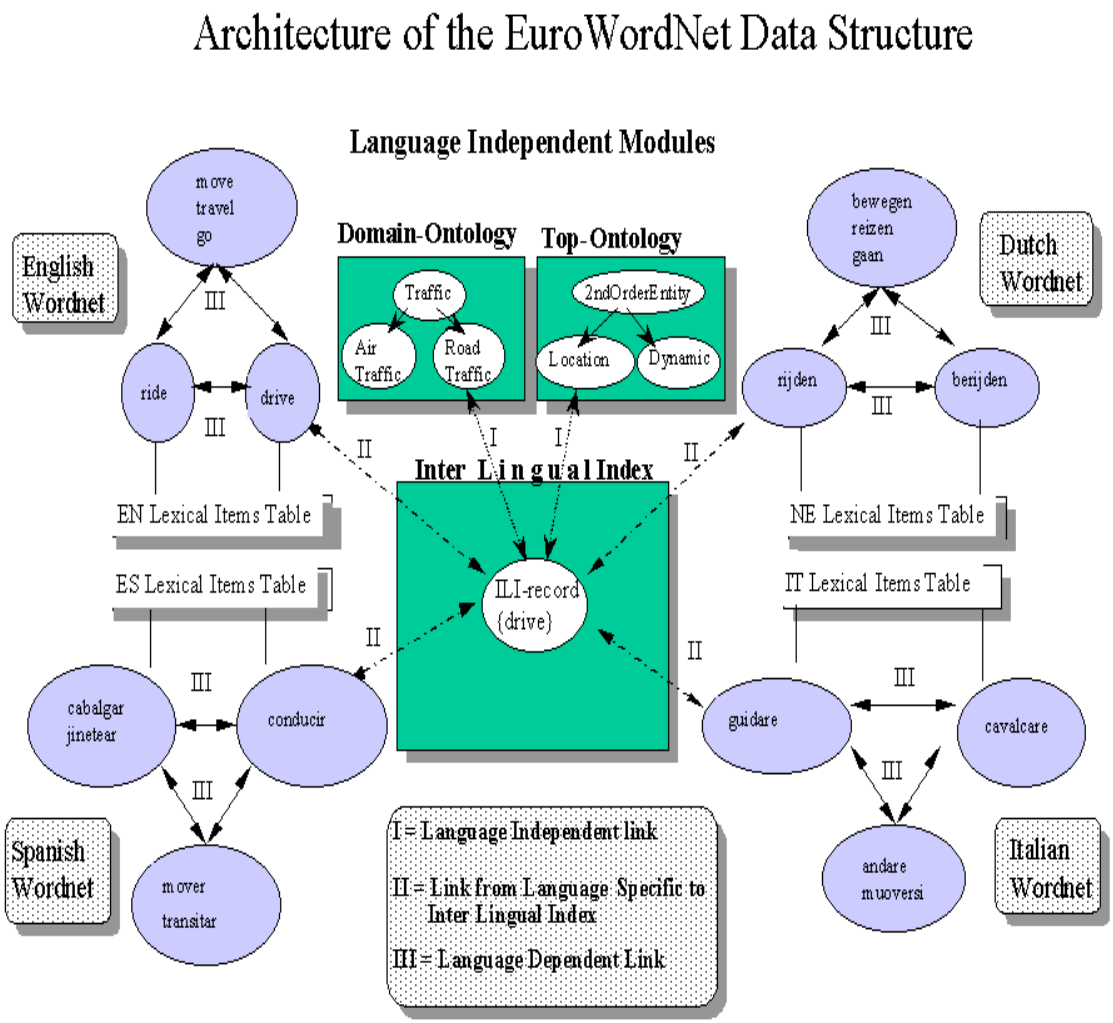


Figura 33: Arquitectura de la estructura de datos de EuroWordNet

4.1. Construcción de la base de datos multilingüe

1. En una primera aproximación general, la construcción de la base de datos pasaría primeramente por reconocer y extraer relaciones semánticas de Diccionarios Electrónicos.

Para cada uno de los idiomas debería crearse la red semántica que relaciona las palabras. Un ejemplo para el caso del wordnet inglés podría ser el siguiente, donde se muestran las relaciones de la palabra “car”:

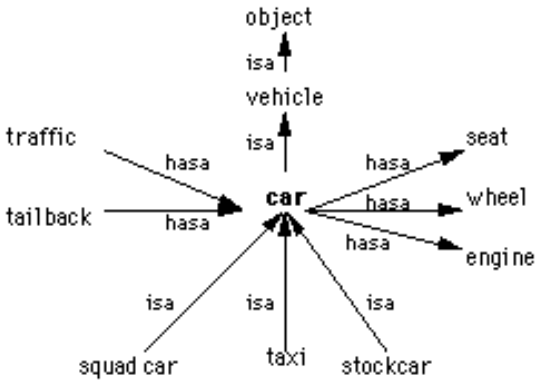


Figura 34: Relaciones para “car”

2. Una vez generada dicha red habríamos obtenido los *synsets* correspondientes, pudiendo entonces unirlos a los *synsets* equivalentes del inglés.

3. Finalmente cargar la red en la base de datos de Novell para procesamiento posterior, comparación de los diferentes *wordnets*, aplicar reglas y pruebas automáticas de coherencia, mezclar y unificar los conceptos más altos en la jerarquía y los conceptos situados en la cima.

4.2. Ejemplo de enlace de palabras de distintos idiomas

Consideremos todas las palabras relacionadas con las “partes del cuerpo”.

Todos los *wornets* compartirán en la ontología el concepto “parte, pero cada lengua tendrá distintas relaciones para las partes del cuerpo.

Mientras que en inglés la palabra “leg” significa lo mismo para los animales y los humanos, en el caso del español, por ejemplo, se emplean distintos términos, tales como “pata” para el caso animal, y “pierna” para las personas.

El caso inverso ocurriría para la palabra “dedo”, ya que en nuestro idioma se usa indistintamente para referirnos a los pies y las manos, no

ocurriendo lo mismo para el inglés donde se separa “toe” y “finger”, respectivamente.

En el esquema siguiente se muestran el idioma inglés, alemán, español, italiano, relacionados a través del índice de significados, en inglés.

Si analizamos el caso del idioma español, se observa que la palabra “dedo”, es indistintamente “finger” o “toe”, por lo que podemos concluir que mantiene una relación de hiponimia con ambos.

Se entiende por hipónimo la palabra cuyo significado es más específico que el de otra en la que está englobada (“finger” es más especifico de “dedo”).

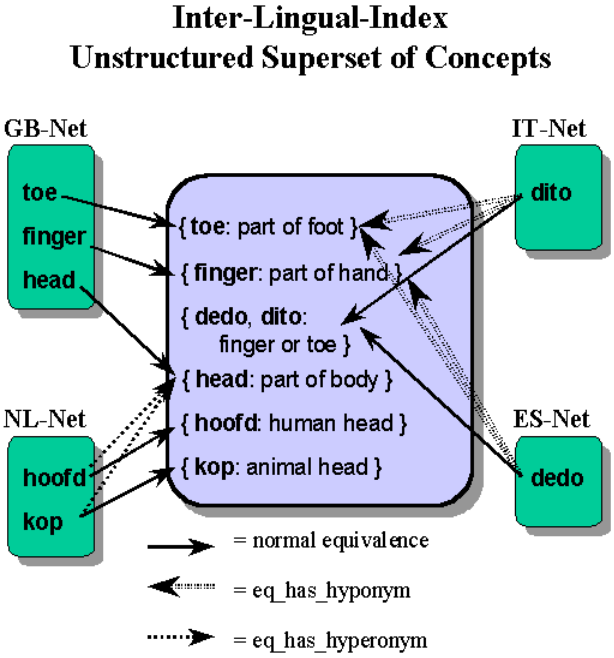


Figura 35: Interlingua de EuroWordNet

5. RELACIONES SEMANTICAS EXISTENTES EN EWN

A continuación se exponen las relaciones que están presentes en EWN para relacionar las palabras.

Para comprenderlas se aclaran algunos significados:

- **Hipónimo**: Palabra cuyo significado está incluido en el de otra; p. ej., *gorrión* respecto a *pájaro*.

- **Metonimia**: Tropo que consiste en designar algo con el nombre de otra cosa tomando el efecto por la causa o viceversa, etc.; p. ej., *las canas* por *la vejez;*

- **Hiperonimia**: Relación de significado de un hiperónimo con respecto a sus hipónimos.

- **Hiperónimo**: Palabra cuyo significado incluye al de otra u otras; p. ej., *pájaro* respecto a *jilguero* y *gorrión*.

relación	ejemplo	descripción
HAS_XPOS_HYPERONYM	destrucción > cambiar	hiperonimia transcategorial
HAS_XPOS_HYPONYM	cambio > destruir	hiponimia transcategorial
HAS_HYPERONYM	destruir > cambiar	hiperonimia
HAS_HYPONYM	cambio > destrucción	hiponimia
XPOS_NEAR_SYNONYM	cambiar > cambio	cuasi-sinonimia transcategorial
NEAR_SYNONYM	cigarro > puro	cuasi-sinonimia
FUZZYNYM	feudalismo > propiedad	relación indeterminada
XPOS_FUZZYNYM	feudalismo > poseer	relación indeterminada transcategorial
ANTONYM	construir > destruir	antonimia
NEAR_ANTONYM	construir > destrozar	cuasi-antonimia
XPOS_NEAR_ANTONYM	construcción > destrozar	cuasi-antonimia transcategorial
INVOLVED	martillear > martillo	entidad directamente relacionada con un evento
ROLE	vino > beber	evento directamente relacionado con una entidad
involved_agent	educar > educador	*involvement en que la entidad realiza un papel agentivo*
role_agent	educador > educar	*role en que la entidad realiza un papel agentivo*
involved_patient	examinar > examinando	*involvement en que la entidad realiza un papel de paciente*
role_patient	examinando > examinar	*role en que la entidad realiza un papel de paciente*
involved_instrument	martillear > martillo	*involvement en que la entidad realiza un papel de instrumento*

role_instrument	martillo > martillear	*role en que la entidad realiza un papel de instrumento*
involved_location	comer > comedor	*involvement en que la entidad realiza un papel locativo*
role_location	comedor > comer	*role en que la entidad realiza un papel locativo*
involved_direction	caer > suelo	*involvement en que la entidad realiza un papel de origen o destino*
involved_source_direction	disparo > arma de fuego	*involvement en que la entidad realiza un papel de origen*
involved_target_direction	caer > suelo	*involvement en que la entidad realiza un papel de destino*
role_direction	suelo > caer	inversa de *involved_direction*
role_source_direction	arma de fuego > disparo	inversa de *involved_source_direction*
role_target_direction	suelo > caer	inversa de *involved_target_direction*
HAS_HOLONYM	nariz > cara	holonimia (genérica)
HAS_MERONYM	cara > nariz	meronimia (genérica)
member	senador > senado	holonimia de inclusión de elemento en conjunto
has_mero_member	senado > senador	inversa de la anterior
portion	mandrugo > pan	holonimia de trozo o fragmento
has_mero_portion	pan > mendrugo	inversa de la anterior
part	nariz > cara	holonimia de parte
has_mero_part	cara > nariz	inversa de la anterior
madeof	papel > libro	holonimia de sustancia, o *hecho de*
has_mero_madeof	libro > papel	inversa de la anterior
location	oasis > desierto	holonimia de lugar
has_mero_location	desierto > oasis	inversa de la anterior
CAUSES	matar > morir	causación entre eventos
IS_CAUSED_BY	morir > matar	inversa de la anterior
HAS_INSTANCE	ciudad > Barcelona	instancia de clase
BELONGS_TO_CLASS	Barcelona > ciudad	inversa de la anterior
BE_IN_STATE	belleza > bello	estado correspondiente a la posesión de una cierta propiedad
STATE_OF	bello > belleza	inversa de la anterior

6. INSTITUTOS ENCARGADO DE ELABORAR LOS WORDNETS

WordNet	**Intituto**
Holandés:	the University of Amsterdam (co-ordinator of EuroWordNet).
Español:	the 'Fundacíon Universidad Empresa' (a co-operation of UNED Madrid, Politecnica de Catalunya in Barcelona, and the University of Barcelona).
Italiano:	Istituto di Linguistica Computazionale, C.N.R., Pisa.
Inglés:	University of Sheffield (adapting the English wordnet).
Francés:	Université d' Avignon and Memodata at Avignon.
Alemán:	Universität Tübingen.
Checo:	University of Masaryk at Brno in Czech.
Estonio:	University of Tartu in Estonia.

Cada uno de estos institutos es el responsable de construir el *wornet* de su país.

7. CONCLUSIONES

Una vez conocido el funcionamiento de EuroWordNet y sus distintos usos, se puede concluir que es un sistema de gran utilidad, por distintas razones, que se exponen a continuación:

- Sirve como punto de partida para la elaboración de un léxico para la Traducción Automática, algo realmente útil en nuestros días.
- Dicho léxico, podría ser la base para desarrollar bases de conocimiento que ofrezcan la posibilidad de creación de sistemas de reconocimiento y comprensión automática del lenguaje.
- Proporciona herramientas de aprendizaje de lenguas, en las que los usuarios que se introducen en una nueva lengua pudieran ojear el vocabulario de la misma a través de la red a partir de una palabra conocida para ellos.

En nuestro caso, al desarrollar un sistema que permita extraer los descriptores que comprende una asignatura a partir de sus programas, un sistema de este tipo es de gran utilidad.

Lo ideal sería que dichos programas fuesen elaborados en un idioma común, normalmente el inglés, facilitando así el trabajo.

Por lo general la diversidad de lenguas en los que se realizan los programas de las asignaturas, donde cada universidad lo confecciona en el idioma de su país, dificulta en gran medida el proceso.

De esta forma si se dispone de un sistema como EuroWordNet, capaz de establecer relaciones entre las palabras de distintos idiomas, esto podría pasar a ser una realidad.

EuroWordNet también presenta algunos inconvenientes, tales como la oferta de lenguas que es capaz de abarcar, ya que en nuestro caso no abarca la totalidad de los que se ofrecen, como por ejemplo portugués.

Actualmente el diseño de la base de datos, la definición de relaciones y la ontología ya no se están desarrollando, aunque existen institutos y grupos de investigación que están trabajando en el desarrollo de otros *Wordnet* para lenguas europeas y no europeas usando como base EuroWordNet, y que pueden ser añadidas a la base de datos y conectarse así con otros idiomas.

C. CÓDIGO

CÓDIGO

1. PROYECTO GICE4S

El código desarrollado como parte de este proyecto se presenta dividido en las clases que son empleadas en cada uno de los apartados en los cuales se ha hecho mención al código del proyecto.

1.1. Apartado V.B. Similitud entre asignaturas

1.1.1. es.ucm.fdi.gice4s.model.Similitud.**DatosOntologia**

```
import java.io.*;

public class DatosOntologia implements Serializable {

  public double[][] tablaSimilitudes;
  public String[] tablaDescriptores;

  public DatosOntologia() {}

  public DatosOntologia(double[][] similitudes, String[] descriptores) {
    tablaSimilitudes = similitudes;
    tablaDescriptores = descriptores;
  }

  public double devuelveSimilitud(String descriptorA, String descriptorB) {
    int posA = devuelvePosicion(descriptorA, tablaDescriptores);
    int posB = devuelvePosicion(descriptorB, tablaDescriptores);
    if((posA > 0) && (posB > 0))
      return tablaSimilitudes[posA][posB];
    else
      return 0;
  }

  private int devuelvePosicion(String descriptor, String[] tabla) {
   int pos = -1;
   int i = 0;
   boolean encontrado = false;
   while(!encontrado && i < tabla.length) {
      if(descriptor.equalsIgnoreCase(tabla[i])) {
         encontrado = true;
         pos = i;
      }
      else
         i++;
   }
   return pos;
  }
}
```

1.1.2. es.ucm.fdi.gice4s.model.Similitud.**Asignatura**

```
import java.util.*;

public class Asignatura {
  private LinkedList listaConvenios;
  private float creditos;
  private String nombre;
  public int caracter;
  public double similitud = 0;
  private int convalidada = 0;
```

```
  private final static int CAR_TR = 0;
  private final static int CAR_OB = 1;
  private final static int CAR_OP = 2;
  private final static int CAR_LC = 3;

  public Asignatura() {}

  public Asignatura(Asignatura asig) {
       listaConvenios = asig.listaConvenios;
       creditos = asig.creditos;
       nombre = asig.nombre;
       caracter = asig.caracter;
       similitud = asig.similitud;
       convalidada = asig.convalidada;
  }

  public Asignatura(String nom, float cred, LinkedList lista) {
    nombre = nom;
    creditos = cred;
    listaConvenios = lista;
  }

  public Asignatura(String nom, float cred, LinkedList lista, int car) {
    nombre = nom;
    creditos = cred;
    listaConvenios = lista;
    caracter = car;
  }

  public LinkedList getListaConvenios() {
    return listaConvenios;
  }

  public float getCreditos() {
    return creditos;
  }

  public String getNombre() {
       return nombre;
  }

  public void setNombre(String nom) {
       nombre = nom;
  }

  public double getSimilitud() {
       return similitud;
  }

  public void setSimilitud(double sim) {
       similitud = sim;
  }

  public int getConvalidada() {
       return convalidada;
  }

  public void setConvalidada(int conv) {
       convalidada = conv;
  }

  public static void main(String[] args) {
    Asignatura asignatura1 = new Asignatura();
  }
}
```

1.1.3. es.ucm.fdi.gice4s.model.Similitud.**Ontologia**

```
import java.io.*;

public class Ontologia {


  private DatosOntologia datosOnto;

  public Ontologia() {
    try {
       java.net.URL urlOnto =
              es.ucm.fdi.gice4s.model.Similitud.Ontologia.class.getResource(
              "ontologia.dat");
       String direccion = urlOnto.getPath();
       direccion = direccion.substring(1);
       datosOnto = (DatosOntologia)abrirFichero(direccion);

    }catch (Exception e1) {
      System.out.println("Exception: " + e1.getMessage());
    }
  }

  public double similitudConvenio(String descriptorA, String descriptorB) {
    return datosOnto.devuelveSimilitud(descriptorA, descriptorB);
  }

  public void guardarFichero(String nombreFichero, Object contenido) {
    try {
      FileOutputStream ostream = new FileOutputStream(nombreFichero);
      ObjectOutputStream archivo = new ObjectOutputStream(ostream);

      archivo.writeObject(contenido);

      archivo.flush();
      ostream.close();
    }
    catch(IOException e1) {
      System.out.println("Error al guardar en el archivo " + nombreFichero);
    }
    catch(Exception e2) {
      System.out.println("Error al guardar en el archivo " + nombreFichero);
    }
  }

  public Object abrirFichero(String nombreFichero) {
     try {
      FileInputStream istream = new FileInputStream(nombreFichero);
      ObjectInputStream archivo = new ObjectInputStream(istream);

      Object obj = archivo.readObject();

      archivo.close();
      istream.close();
      return obj;
    }
    catch(IOException e1) {
      System.out.println("Error al abrir el archivo " + nombreFichero);
      System.out.println(e1.getMessage());
      return null;
    }
    catch(Exception e2) {
      System.out.println("Error al abrir el archivo " + nombreFichero);
      System.out.println(e2.getMessage());
      return null;
    }
  }
```

```
  public static void main(String[] args) {}
}
```

1.1.4. es.ucm.fdi.gice4s.model.Similitud.**SimilitudAsignatura**

```
import java.util.*;

public class SimilitudAsignaturas {

  private static Ontologia ontologia;

  public SimilitudAsignaturas() {
    ontologia = new Ontologia();
  }

  public double similitudAsignatura(Asignatura[] asignaturasOrigen,
                  Asignatura[] asignaturasDestino) {
    LinkedList listaConveniosOrigen = new LinkedList();
    float creditosOrigen = 0;
    for(int i = 0; i < asignaturasOrigen.length; i++) {
      LinkedList conveniosOrigen = asignaturasOrigen[i].getListaConvenios();
      creditosOrigen += asignaturasOrigen[i].getCreditos();
      for(int j = 0; j < conveniosOrigen.size(); j++) {
        if(!listaConveniosOrigen.contains((String)conveniosOrigen.get(j))) {
          listaConveniosOrigen.add((String)conveniosOrigen.get(j));
        }
      }
    }
    LinkedList listaConveniosDestino = new LinkedList();
    float creditosDestino = 0;
    for(int i = 0; i < asignaturasDestino.length; i++) {
      LinkedList conveniosDestino = asignaturasDestino[i].getListaConvenios();
      creditosDestino += asignaturasDestino[i].getCreditos();
      for(int j = 0; j < conveniosDestino.size(); j++) {
        if(!listaConveniosDestino.contains((String)conveniosDestino.get(j))) {
          listaConveniosDestino.add((String)conveniosDestino.get(j));
        }
      }
    }
    Asignatura asignaturaOrigen = new Asignatura("Asignaturas Origen",
       creditosOrigen, listaConveniosOrigen);
    Asignatura asignaturaDestino = new Asignatura("Asignaturas Destino",
       creditosDestino, listaConveniosDestino);
    double similitud =similitudAsignatura(asignaturaOrigen,asignaturaDestino);
    return similitud;
  }

  public double similitudAsignatura(Asignatura asigOrigen, Asignatura
                                       asigDestino) {
      try {
            LinkedList conveniosOrigen = new LinkedList();
            LinkedList conveniosDestino = new LinkedList();
            conveniosOrigen =
                  (LinkedList)asigOrigen.getListaConvenios().clone();
            conveniosDestino =
                  (LinkedList)asigDestino.getListaConvenios().clone();
            double simContenido = similitudContenido(conveniosOrigen,
                  conveniosDestino);
            double simCreditos = similitudCreditos(asigOrigen.getCreditos(),
                  asigDestino.getCreditos());
            return (simContenido * simCreditos);
      } catch (Exception e){
            System.out.println(e.getMessage());
            e.printStackTrace();
            return 0;
```

```
    }
}

public double similitudContenido(LinkedList conveniosOrigen, LinkedList
                                  conveniosDestino) {
  int longConvOrigen = conveniosOrigen.size();
  int longConvDestino = conveniosDestino.size();
  // Comparamos los que sean iguales
  int numIguales = 0;
  for(int i = 0; i < conveniosOrigen.size(); i++) {
    String convenio = (String)conveniosOrigen.get(i);
    for(int j = 0; j < conveniosDestino.size(); j++) {
      String convenioDestino = (String)conveniosDestino.get(j);
      if(convenio.equals(convenioDestino)) {
        numIguales++;
        conveniosDestino.remove(j); j--;
        conveniosOrigen.remove(i); i--;
      }
    }
  }
  LinkedList tablaSimilitudes = new LinkedList();
  // Comparamos los que no sean iguales
  //Creamos la tabla de similitudes
  for(int i = 0; i < conveniosOrigen.size(); i++) {
    String convenio = (String)conveniosOrigen.get(i);
    LinkedList filaSimilitudes;
    if(conveniosDestino.size() != 0) {
     filaSimilitudes = new LinkedList();
     for(int j = 0; j < conveniosDestino.size(); j++) {
            String convenioDestino = (String)conveniosDestino.get(j);
            double sim = ontologia.similitudConvenio(convenio,
                   convenioDestino);
            filaSimilitudes.addLast(new Double(sim));
     }
     tablaSimilitudes.addLast(filaSimilitudes);
    }
  }
  //Los cogemos de mayor a menor parecido
  LinkedList similitudes = new LinkedList();
  int tamTabla = tablaSimilitudes.size();
  for(int i = 0; i < tamTabla; i++) {
    double max = 0;
    int filaMax = 0;
    int columnaMax = 0;
    for(int j = 0; j < tablaSimilitudes.size(); j++) {
      int tamFila = ((LinkedList)tablaSimilitudes.get(j)).size();
      for(int k = 0; k < tamFila; k++) {
        double celda = ((Double)((LinkedList)
                  tablaSimilitudes.get(j)).get(k)).doubleValue();
        if(celda > max) {
          max = celda;
          filaMax = j;
          columnaMax = k;
        }
      }
    }
    //Quitamos la fila y la columna del máximo
    similitudes.addLast(new Double(max));
    tablaSimilitudes.remove(filaMax);
    for(int l = 0; l < tablaSimilitudes.size(); l++) {
     if(((LinkedList)tablaSimilitudes.get(l)).size() != 0)
            ((LinkedList)tablaSimilitudes.get(l)).remove(columnaMax);
    }
  }
  // Combinamos las similitudes parciales obtenidas
  double similitudDistintos = 1;
  for(int i = 0; i < similitudes.size(); i++) {
    similitudDistintos =
```

```
            similitudDistintos * ((Double)similitudes.get(i)).doubleValue();
    }
    double similitud = 0;
    if (conveniosOrigen.size() != 0) {
       // Combinamos las similitudes de los distintos y los iguales
        similitud = (similitudDistintos * 0.2) +
                    ((numIguales / (double)longConvOrigen) * 0.8);
    }
    else
       similitud = numIguales / (double)longConvDestino;

    return similitud;
  }

  public static double similitudCreditos(float creditosOrigen, float
                                         creditosDestino) {
    double similitud = 0;
    if(creditosOrigen <= creditosDestino)
      similitud = 1;
    else
      similitud = (creditosDestino / creditosOrigen);

    return similitud;
  }

   public static void main(String [] args) {}

}
```

1.2 Apartado V.C. Sistema de ayuda al alumno para la elección de la universidad destino

1.2.1 es.ucm.fdi.gice4s.model.Preferencias.**EntradaBD**

```
import java.io.Serializable;
import java.sql.*;

public class EntradaBD implements Serializable {

  public String language;
  public String country;
  private String name;
  public int distanceCountry;
  public int distanceCity;
  private double puntuacion = 0;
  private double puntPref = 0;
  private double puntAsig = 0;

  public EntradaBD() {}

  public EntradaBD(String lan, String cou, String nam, int dCo, int dCi) {
       language = lan;
       country = cou;
       name = nam;
       distanceCountry = dCo;
       distanceCity = dCi;
       puntuacion = 0;
  }

  public EntradaBD(ResultSet rset) throws SQLException {
       language = (String) rset.getObject("Language");
       Integer distance_c = (Integer)
       rset.getObject("DistanceCountry");
       distanceCountry = distance_c.intValue();
       Integer distance_ci=(Integer)rset.getObject("DistanceCity");
```

```
        distanceCity = distance_ci.intValue();
        country = (String) rset.getObject("Country");
        name = (String) rset.getObject("Name");
        puntuacion = 0;
    }

    public String toString() {
      int puntos_int = (int)(puntuacion * 100);
      String cadena = "(" + puntos_int + " ptos) Idioma " + language + "  .
         Distancia al pais  " + distanceCountry + "    .Distancia a la     Ciudad
         " + distanceCity + "Pais de destino" + country ;
      return cadena;
    }

    public String getName(){
        return name;
    }

    public void setName(String name){
        this.name = name;
    }

    public double getPuntuacion(){
        return puntuacion;
    }

    public void setPuntuacion(double punt){
        this.puntuacion = punt;
    }

    public double getPuntuacionAsig(){
        return puntAsig;
    }

    public void setPuntuacionAsig(double punt){
        this.puntAsig = punt;
    }

    public double getPuntuacionPref(){
        return puntPref;
    }

    public void setPuntuacionPref(double punt){
        this.puntPref = punt;
    }

    public static void main(String[] args) {
      EntradaBD entradaBD1 = new EntradaBD();
    }
}
```

1.2.2 es.ucm.fdi.gice4s.model.Preferencias.**Similitud**

```
public class Similitud {

    //Idioma
    public static final int INGLES = 0;
    public static final int FRANCES = 1;
    public static final int PORTUGUES = 2;
    public static final int ITALIANO = 3;
    public static final int ALEMAN = 4;

    //Country
    public static final int INGLATERRA = 0;
    public static final int CHIPRE = 1;
    public static final int FRANCIA = 2;
```

```
public static final int ALEMANIA = 3;
public static final int AUSTRIA = 4;
public static final int POLONIA = 5;
public static final int BELGICA = 6;
public static final int PORTUGAL = 7;
public static final int ITALIA = 8;
public static final int FINLANDIA = 9;
public static final int HOLANDA = 10;

//Tablas
private static double[][] tablaLanguage = new double[5][5];
private static double[][] tablaCountry = new double[11][11];

public Similitud() {}

private static void crearDatosLanguage() {
  //Tipo de turismo
  tablaLanguage[INGLES][INGLES] = 1.0;
  tablaLanguage[INGLES][FRANCES] = 0.3;
  tablaLanguage[INGLES][PORTUGUES] = 0.3;
  tablaLanguage[INGLES][ITALIANO] = 0.2;
  tablaLanguage[INGLES][ALEMAN] = 0.5;

  tablaLanguage[FRANCES][FRANCES] = 1.0;
  tablaLanguage[FRANCES][PORTUGUES] = 0.3;
  tablaLanguage[FRANCES][ITALIANO] = 0.5;
  tablaLanguage[FRANCES][ALEMAN] = 0.2;

  tablaLanguage[PORTUGUES][PORTUGUES] = 1.0;
  tablaLanguage[PORTUGUES][ITALIANO] = 0.5;
  tablaLanguage[PORTUGUES][ALEMAN] = 0.1;

  tablaLanguage[ITALIANO][ITALIANO] = 1.0;
  tablaLanguage[ITALIANO][ALEMAN] = 0.1;

  tablaLanguage[ALEMAN][ALEMAN] = 1.0;
}

private static void crearDatosAlojamiento() {
  //Alojamiento
  tablaCountry[INGLATERRA][INGLATERRA] = 1.0;
  tablaCountry[INGLATERRA][CHIPRE] = 0.4;
  tablaCountry[INGLATERRA][FRANCIA] = 0.5;
  tablaCountry[INGLATERRA][ALEMANIA] = 0.4;
  tablaCountry[INGLATERRA][AUSTRIA] = 0.4;
  tablaCountry[INGLATERRA][POLONIA] = 0.2;
  tablaCountry[INGLATERRA][BELGICA] = 0.3;
  tablaCountry[INGLATERRA][PORTUGAL] = 0.3;
  tablaCountry[INGLATERRA][ITALIA] = 0.4;
  tablaCountry[INGLATERRA][FINLANDIA] = 0.2;
  tablaCountry[INGLATERRA][HOLANDA] = 0.2;


  tablaCountry[CHIPRE][CHIPRE] = 1.0;
  tablaCountry[CHIPRE][FRANCIA] = 0.4;
  tablaCountry[CHIPRE][ALEMANIA] = 0.3;
  tablaCountry[CHIPRE][AUSTRIA] = 0.3;
  tablaCountry[CHIPRE][POLONIA] = 0.3;
  tablaCountry[CHIPRE][BELGICA] = 0.4;
  tablaCountry[CHIPRE][PORTUGAL] = 0.3;
  tablaCountry[CHIPRE][ITALIA] = 0.3;
  tablaCountry[CHIPRE][FINLANDIA] = 0.3;
  tablaCountry[CHIPRE][HOLANDA] = 0.3;

  tablaCountry[FRANCIA][FRANCIA] = 1.0;
  tablaCountry[FRANCIA][PORTUGAL] = 0.2;
  tablaCountry[FRANCIA][AUSTRIA] = 0.2;
  tablaCountry[FRANCIA][POLONIA] = 0.3;
```

```
  tablaCountry[FRANCIA][BELGICA] = 0.6;
  tablaCountry[FRANCIA][PORTUGAL] = 0.2;
  tablaCountry[FRANCIA][ITALIA] = 0.2;
  tablaCountry[FRANCIA][FINLANDIA] = 0.3;
  tablaCountry[FRANCIA][HOLANDA] = 0.5;

  tablaCountry[ALEMANIA][ALEMANIA] = 1.0;
  tablaCountry[ALEMANIA][AUSTRIA] = 0.7;
  tablaCountry[ALEMANIA][POLONIA] = 0.4;
  tablaCountry[ALEMANIA][BELGICA] = 0.3;
  tablaCountry[ALEMANIA][PORTUGAL] = 0.2;
  tablaCountry[ALEMANIA][ITALIA] = 0.2;
  tablaCountry[ALEMANIA][FINLANDIA] = 0.3;
  tablaCountry[ALEMANIA][HOLANDA] = 0.3;

  tablaCountry[AUSTRIA][AUSTRIA] = 1.0;
  tablaCountry[AUSTRIA][POLONIA] = 0.2;
  tablaCountry[AUSTRIA][BELGICA] = 0.3;
  tablaCountry[AUSTRIA][PORTUGAL] = 0.3;
  tablaCountry[AUSTRIA][ITALIA] = 0.3;
  tablaCountry[AUSTRIA][FINLANDIA] = 0.2;
  tablaCountry[AUSTRIA][HOLANDA] = 0.2;

  tablaCountry[POLONIA][POLONIA] = 1.0;
  tablaCountry[POLONIA][BELGICA] = 0.3;
  tablaCountry[POLONIA][PORTUGAL] = 0.3;
  tablaCountry[POLONIA][ITALIA] = 0.3;
  tablaCountry[POLONIA][FINLANDIA] = 0.2;
  tablaCountry[POLONIA][HOLANDA] = 0.2;

  tablaCountry[BELGICA][BELGICA] = 1.0;
  tablaCountry[BELGICA][PORTUGAL] = 0.2;
  tablaCountry[BELGICA][ITALIA] = 0.2;
  tablaCountry[BELGICA][FINLANDIA] = 0.2;
  tablaCountry[BELGICA][HOLANDA] = 0.5;

  tablaCountry[PORTUGAL][PORTUGAL] = 1.0;
  tablaCountry[PORTUGAL][ITALIA] = 0.6;
  tablaCountry[PORTUGAL][FINLANDIA] = 0.1;
  tablaCountry[PORTUGAL][HOLANDA] = 0.1;

  tablaCountry[ITALIA][ITALIA] = 1.0;
  tablaCountry[ITALIA][FINLANDIA] = 0.2;
  tablaCountry[ITALIA][HOLANDA] = 0.2;

  tablaCountry[FINLANDIA][FINLANDIA] = 1.0;
  tablaCountry[FINLANDIA][HOLANDA] = 0.2;

  tablaCountry[HOLANDA][HOLANDA] = 1.0;
}

public static double getPuntosLanguage(int x, int y) {
  crearDatosLanguage();
  if (y < x) {
    int aux = y;
    y = x;
    x = aux;
  }
  return tablaLanguage[x][y];
}

public static double getPuntosCountry(int x, int y) {
  crearDatosAlojamiento();
  if (y < x) {
    int aux = y;
    y = x;
    x = aux;
  }
```

```
    return tablaCountry[x][y];
  }

  public static void main(String[] args) {
    Similitud similitud1 = new Similitud();
  }
}
```

1.2.3. es.ucm.fdi.gice4s.model.Preferencias.**Eleccion**

```
import java.io.Serializable;

public class Eleccion implements Serializable {

	public static final int E_IMPRESCINDIBLE = 4;
	public static final int E_IMPORTANTE = 3;
	public static final int E_RELEVANTE = 2;
	public static final int E_PREFERIBLE = 1;
	public static final int E_NO_RELEVANTE = 0;
	private String campo = "";
	private String valor = "";
	private int importancia = E_NO_RELEVANTE;

	public Eleccion() {}

	public Eleccion(String c, String v, int imp) {
		campo = c;
		valor = v;
		importancia = imp;
	}

	public String getValor() {
		return valor;
	}

	public void setValor(String v) {
		valor = v;
	}

	public String getCampo() {
		return campo;
	}

	public void setCampo(String c) {
		campo = c;
	}

	public int getImportancia() {
		return importancia;
	}

	public void setImportancia(int imp) {
		importancia = imp;
	}

	public static void main(String[] args) {
		 Eleccion eleccion1 = new Eleccion();
	}
}
```

1.2.4 es.ucm.fdi.gice4s.model.Preferencias.**Inteligencia**

```
import java.io.IOException;
import java.util.*;
import javax.servlet.ServletException;
import es.ucm.fdi.gice4s.http.controller.session.DataManager;
import es.ucm.fdi.gice4s.util.exceptions.InternalErrorException;
```

```
public class Inteligencia {

	LinkedList listaElecciones = new LinkedList();
	LinkedList listaPosibles = new LinkedList();
	LinkedList listaMejores = new LinkedList();
	double maxPuntuacion = 0;

	public Inteligencia() {}

	public Inteligencia(LinkedList listaE) {
		listaElecciones = listaE;
	}

	public LinkedList devuelveListaMejores()throws IOException,
			ServletException, InternalErrorException {

		obtenerMaximaPuntuacion();
		obtenerPosibles();
		obtenerMejores();
		for(int i = 0; i < listaMejores.size(); i++) {
			EntradaBD entrada = (EntradaBD)listaMejores.get(i);
			double puntuacion = entrada.getPuntuacion();
			if (maxPuntuacion != 0)
				entrada.setPuntuacion(puntuacion / maxPuntuacion);
			else
				entrada.setPuntuacion(-1);
			listaMejores.set(i, entrada);
		}
		return listaMejores;
	}

	public void obtenerPosibles() throws IOException, ServletException,
				InternalErrorException{

		/*Seleccionamos de la lista de posibles solo los que sean
		imprescindibles, para hacer un filtrado exacto con esos valores
		de la  BBDD*/
		LinkedList listaImprescindibles = new LinkedList();
		ArrayList listaPosiblesAux = new ArrayList();
		int importancia = 0;
		for (int i = 0; i < listaElecciones.size(); i++) {
			Eleccion eleccion = ( (Eleccion) listaElecciones.get(i));
			importancia = eleccion.getImportancia();
			if (importancia == Eleccion.E_IMPRESCINDIBLE) {
				listaImprescindibles.addLast(eleccion);
			}
		}
		listaPosiblesAux =
		(ArrayList)DataManager.obtenerPosibles(listaImprescindibles);
		for(int i = 0; i< ((ArrayList)listaPosiblesAux.get(0)).size();
		i++){
			String language =
			(String)((ArrayList)listaPosiblesAux.get(0)).get(i);
			String country =
			(String)((ArrayList)listaPosiblesAux.get(1)).get(i);
			String name =
			(String)((ArrayList)listaPosiblesAux.get(2)).get(i);
			int countryDistance =
			Integer.parseInt(((String)((ArrayList)listaPosiblesAux.get(
				3)).get(i)));
			int cityDistance =
			Integer.parseInt((String)((ArrayList)listaPosiblesAux.get(4
			)).get(i));
			EntradaBD entrada = new EntradaBD(language, country, name,
			countryDistance, cityDistance);
			listaPosibles.addLast(entrada);
		}
```

```
    }

    private void obtenerMejores() {
           double puntuacion = 0;
           EntradaBD entrada;
           LinkedList listaCamposConsiderados = new LinkedList();
           for (int i = 0; i < listaElecciones.size(); i++) {
                  Eleccion eleccion = ( (Eleccion) listaElecciones.get(i));
                  int importancia = eleccion.getImportancia();
                  if (importancia != Eleccion.E_IMPRESCINDIBLE) {
                         listaCamposConsiderados.addLast(eleccion);
                  }
           }
           try {
                  for (int i = 0; i < listaPosibles.size(); i++) {
                         entrada = ( (EntradaBD) listaPosibles.get(i));
                         puntuacion = darPuntuacion(entrada,
                         listaCamposConsiderados);
                         insertaOrden(puntuacion, entrada);
                  }
           } catch(Exception e) {
                  System.out.println(e.getMessage());
           }
    }

private double darPuntuacion(EntradaBD entrada, LinkedList lista) {
  double puntuacion = 0;
  Eleccion eleccion;
  for (int i = 0; i < lista.size(); i++) {
    eleccion = ( (Eleccion) lista.get(i));
    if (eleccion.getCampo().equals("Language")) {
      puntuacion +=
          darPuntuacionLanguage(entrada.language, eleccion.getValor(),
                                eleccion.getImportancia());
    }
    else if (eleccion.getCampo().equals("Country")) {
      puntuacion +=
          darPuntuacionCountry(entrada.country, eleccion.getValor(),
                               eleccion.getImportancia());
    }
    else if (eleccion.getCampo().equals("DistanceCity")) {
      puntuacion +=
          darPuntuacionDistanceCity(entrada.distanceCity,
                  eleccion.getValor(),eleccion.getImportancia());
    }
    else if (eleccion.getCampo().equals("DistanceCountry")) {
      puntuacion +=
          darPuntuacionDistanceCountry(entrada.distanceCountry,
                  eleccion.getValor(),eleccion.getImportancia());
    }
    else {
      System.out.println("ERROR (darPuntuacion): Campo \"" +
                         eleccion.getCampo() + "\" desconocido");
    }
  }
  entrada.setPuntuacion(puntuacion);
  return puntuacion;
}

private double darPuntuacionLanguage(String valor_entrada,
                                     String valor_eleccion,
                                     int importancia) {
  //Pasamos valor_entrada a un entero
  int valor_entrada_int = 0;
  if (valor_entrada.equals("inglés")) {
    valor_entrada_int = Similitud.INGLES;
  }
  else if (valor_entrada.equals("francés")) {
```

```
    valor_entrada_int = Similitud.FRANCES;
  }
  else if (valor_entrada.equals("portugués")) {
    valor_entrada_int = Similitud.PORTUGUES;
  }
  else if (valor_entrada.equals("alemán")) {
    valor_entrada_int = Similitud.ALEMAN;
  }
  else if (valor_entrada.equals("italiano")) {
    valor_entrada_int = Similitud.ITALIANO;
  }
  //Pasamos valor_eleccion a un entero
  int valor_eleccion_int = 0;
  if (valor_eleccion.equals("inglés")) {
    valor_eleccion_int = Similitud.INGLES;
  }
  else if (valor_eleccion.equals("francés")) {
    valor_eleccion_int = Similitud.FRANCES;
  }
  else if (valor_eleccion.equals("alemán")) {
    valor_eleccion_int = Similitud.ALEMAN;
  }
  else if (valor_eleccion.equals("portugués")) {
    valor_eleccion_int = Similitud.PORTUGUES;
  }
  else if (valor_eleccion.equals("italiano")) {
    valor_eleccion_int = Similitud.ITALIANO;
  }
  //Consultamos la tabla de similitudes
  double puntos = Similitud.getPuntosLanguage(valor_entrada_int,
                                              valor_eleccion_int);
  //Hacemos la ponderación
  double ponderacion = obtenerPonderacion(importancia);
  return (puntos * ponderacion);
}

private double darPuntuacionCountry(String valor_entrada,
                                    String valor_eleccion,
                                    int importancia) {
  //Pasamos valor_entrada a un entero
  int valor_entrada_int = 0;
  if (valor_entrada.equals("Inglaterra")) {
    valor_entrada_int = Similitud.INGLATERRA;
  }
  else if (valor_entrada.equals("Chipre")) {
    valor_entrada_int = Similitud.CHIPRE;
  }
  else if (valor_entrada.equals("Francia")) {
    valor_entrada_int = Similitud.FRANCIA;
  }
  else if (valor_entrada.equals("Alemania")) {
    valor_entrada_int = Similitud.ALEMANIA;
  }
  else if (valor_entrada.equals("Austria")) {
    valor_entrada_int = Similitud.AUSTRIA;
  }
  else if (valor_entrada.equals("Polonia")) {
    valor_entrada_int = Similitud.POLONIA;
  }
  else if (valor_entrada.equals("Bélgica")) {
    valor_entrada_int = Similitud.BELGICA;
  }
  else if (valor_entrada.equals("Portugal")) {
    valor_entrada_int = Similitud.PORTUGAL;
  }
  else if (valor_entrada.equals("Finlandia")) {
      valor_entrada_int = Similitud.FINLANDIA;
  }
```

```
  else if (valor_entrada.equals("Holanda")) {
      valor_entrada_int = Similitud.HOLANDA;
  }
  else if (valor_entrada.equals("Italia")) {
      valor_entrada_int = Similitud.ITALIA;
  }
  //Pasamos valor_eleccion a un entero
  int valor_eleccion_int = 0;
  if (valor_eleccion.equals("Inglaterra")) {
    valor_eleccion_int = Similitud.INGLATERRA;
  }
  else if (valor_eleccion.equals("Chipre")) {
    valor_eleccion_int = Similitud.CHIPRE;
  }
  else if (valor_eleccion.equals("Francia")) {
    valor_eleccion_int = Similitud.FRANCIA;
  }
  else if (valor_eleccion.equals("Alemania")) {
    valor_eleccion_int = Similitud.ALEMANIA;
  }
  else if (valor_eleccion.equals("Austria")) {
    valor_eleccion_int = Similitud.AUSTRIA;
  }
  else if (valor_eleccion.equals("Polonia")) {
    valor_eleccion_int = Similitud.POLONIA;
  }
  else if (valor_eleccion.equals("Bélgica")) {
    valor_eleccion_int = Similitud.BELGICA;
  }
  else if (valor_eleccion.equals("Portugal")) {
    valor_eleccion_int = Similitud.PORTUGAL;
  }
  else if (valor_eleccion.equals("Finlandia")) {
      valor_eleccion_int = Similitud.FINLANDIA;
    }
  else if (valor_eleccion.equals("Holanda")) {
      valor_eleccion_int = Similitud.HOLANDA;
    }
  else if (valor_eleccion.equals("Italia")) {
      valor_eleccion_int = Similitud.ITALIA;
    }

  //Consultamos la tabla de similitudes
  double puntos = Similitud.getPuntosCountry(valor_entrada_int,
      valor_eleccion_int);
  //Hacemos la ponderación
  double ponderacion = obtenerPonderacion(importancia);
  return (puntos * ponderacion);
}

private double darPuntuacionDistanceCountry(int valor_entrada, String
                                    valor_eleccion,int importancia) {
  double puntos = 0;
  double longitud = 2900;
  // El usuario ha elegido "Menos de 1000km"
  if (valor_eleccion.equals("Cerca (<1000km)")) {
    if (valor_entrada < 1000) {
      puntos = 1.0;
    }
    else {
      puntos = 1.0 - (valor_entrada - 1000) / longitud;
    }
  }
  // El usuario ha elegido entre 1000 y 2000km
  else if (valor_eleccion.equals("Lejos (1000km - 2000km)")) {
    if ( (valor_entrada >= 1000) && (valor_entrada <= 2000)) {
      puntos = 1.0;
    }
```

```
    else {
      puntos = 1.0 - Math.abs(valor_entrada - 1500) / longitud;
    }
  }
  // El usuario ha elegido mas de 2000km
  else if (valor_eleccion.equals("Muy Lejos (>2000km)")) {
    if (valor_entrada > 2000) {
      puntos = 1.0;
    }
    else {
      puntos = 1.0 - Math.abs(valor_entrada - 2000) / longitud;
    }
  }

  //Hacemos la ponderación
  double ponderacion = obtenerPonderacion(importancia);
  return (puntos * ponderacion);
}
private double darPuntuacionDistanceCity(int valor_entrada, String
                                    valor_eleccion,int importancia) {
  double puntos = 0;
  double longitud = 700;
  // El usuario ha elegido "Menos de 1000km"
  if (valor_eleccion.equals("Menos de 10km")) {
    if (valor_entrada < 10) {
      puntos = 1.0;
    }
    else {
      puntos = 1.0 - (valor_entrada - 10) / longitud;
    }
  }
  // El usuario ha elegido entre 1000 y 2000km
  else if (valor_eleccion.equals("Entre 10km y 20km")) {
    if ( (valor_entrada >= 10) && (valor_entrada <= 20)) {
      puntos = 1.0;
    }
    else {
      puntos = 1.0 - Math.abs(valor_entrada - 15) / longitud;
    }
  }
  // El usuario ha elegido mas de 2000km
  else if (valor_eleccion.equals("Mas de 20km")) {
    if (valor_entrada > 20) {
      puntos = 1.0;
    }
    else {
      puntos = 1.0 - Math.abs(valor_entrada - 20) / longitud;
    }
  }

  //Hacemos la ponderación
  double ponderacion = obtenerPonderacion(importancia);
  return (puntos * ponderacion);
}


private void insertaOrden(double puntuacion, EntradaBD entrada) {
  int i = 0;
  boolean encontrado = false;
  while ( (i < listaMejores.size()) && !encontrado) {
    if (puntuacion > ( (EntradaBD) listaMejores.get(i)).getPuntuacion()) {
      listaMejores.add(i, entrada);
      encontrado = true;
    }
    i++;
  }
  if(!encontrado)
    listaMejores.addLast(entrada);
```

```
  }

  private double obtenerPonderacion(int importancia) {
    switch (importancia) {
      case Eleccion.E_NO_RELEVANTE:
       return 0.0;
      case Eleccion.E_PREFERIBLE:
        return 0.3;
      case Eleccion.E_RELEVANTE:
       return 0.7;
      case Eleccion.E_IMPORTANTE:
        return 1.0;
      case Eleccion.E_IMPRESCINDIBLE:
        System.out.println(
            "WARNING (Inteligencia.obtenerPonderacion): Importancia
                    inesperada");
        return 1.0;
      default:
        System.out.println(
            "ERROR (Inteligencia.obtenerPonderacion): Importancia desconocida"
                    + importancia);
        return -1;
    }
  }

  private void obtenerMaximaPuntuacion() {
       for(int i = 0; i < listaElecciones.size(); i++) {
             int importancia =
                    ((Eleccion)listaElecciones.get(i)).getImportancia();
             switch (importancia) {
                    case Eleccion.E_NO_RELEVANTE:
                           maxPuntuacion = maxPuntuacion + 0.0;
                           break;
                    case Eleccion.E_PREFERIBLE:
                           maxPuntuacion = maxPuntuacion + 0.3;
                           break;
                    case Eleccion.E_RELEVANTE:
                           maxPuntuacion = maxPuntuacion + 0.7;
                           break;
                    case Eleccion.E_IMPORTANTE:
                           maxPuntuacion = maxPuntuacion + 1.0;
                           break;
                    case Eleccion.E_IMPRESCINDIBLE:
                           System.out.println(
                                  "WARNING (Inteligencia.obtenerPonderacion):
                                                Importancia inesperada");
                           maxPuntuacion = maxPuntuacion + 1.0;
                           break;
                    default:
                           System.out.println(
                                  "ERROR (Inteligencia.obtenerPonderacion):
                                  Importancia desconocida" + importancia);
             }
       }
  }
}
```

1.2.5. es.ucm.fdi.gice4s.http.controller.actions.studentout. **InsertParamPreferenciasFormAction**

```
import java.io.IOException;
import java.util.ArrayList;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import org.apache.struts.action.ActionForm;
import org.apache.struts.action.ActionForward;
```

```
import org.apache.struts.action.ActionMapping;
import es.ucm.fdi.gice4s.http.controller.session.DataManager;
import es.ucm.fdi.gice4s.http.view.actionforms.studentout.PreferenciasForm;
import es.ucm.fdi.gice4s.util.exceptions.InternalErrorException;
import es.ucm.fdi.gice4s.util.struts.action.DefaultAction;
import es.ucm.fdi.gice4s.http.view.applicationobjects.*;

public class InsertParamPreferenciasFormAction extends DefaultAction
{
    protected ActionForward doExecute(ActionMapping mapping, ActionForm form,
             HttpServletRequest request, HttpServletResponse response) throws
             IOException, ServletException, InternalErrorException
    {
        PreferenciasForm pef = (PreferenciasForm)form;

        /*Insertamos en el formulario los idiomas*/
        ArrayList formLanguages = new ArrayList();
        ArrayList languages = new ArrayList(DataManager.getLanguages());
        for( int i=0; i<languages.size(); i++) {
            String language = (String)languages.get(i);
            formLanguages.add(new Study(i+1,language));
        }
        pef.setLanguages(formLanguages);

        int[] sel =  new int[1];
        sel[0]=4;
        pef.setSelectedLanguages(sel);
       /*Insertamos en el formulario los Pidiomas*/
        ArrayList formPreferencias1 = new ArrayList();
        String[] preferencias = pef.getPreferencias();
        for( int i=0; i<preferencias.length; i++) {
            if(!preferencias[i].equals("Imprescindible")){
             String pref = (String)preferencias[i];
                formPreferencias1.add(new Study(i+1,pref));
            }
        }
        pef.setPLanguages(formPreferencias1);
        pef.setSelectedPLanguages(sel);

        /*Insertamos en el formulario los paises*/
        ArrayList formCountries = new ArrayList();
        ArrayList countries = new ArrayList(DataManager.getCountries());
        for( int i=0; i<countries.size(); i++) {
            String country = (String)countries.get(i);
            formCountries.add(new Study(i+1,country));
        }
        pef.setCountries(formCountries);
        pef.setSelectedCountries(sel);
        /*Insertamos en el formulario los Ppaises*/
        ArrayList formPreferencias = new ArrayList();
        for( int i=0; i<preferencias.length; i++) {
           String pref = (String)preferencias[i];
           formPreferencias.add(new Study(i+1,pref));
        }
        pef.setPCountries(formPreferencias);
        pef.setSelectedPCountries(sel);

        /*Insertamos en el formulario las distancias al pais*/
        ArrayList formCountryDistance = new ArrayList();
        //ArrayList countryDistance = new
             ArrayList(pef.getCountriesDistance());
        String[] countryDistance = pef.getCountriesDistance();
        for( int i=0; i<countryDistance.length; i++) {
            String pref = (String)countryDistance[i];
            formCountryDistance.add(new Study(i+1,pref));
        }
        pef.setCountryDistance(formCountryDistance);
        pef.setSelectedCountryDistance(sel);
```

```
        /*Insertamos en el formulario las Pdistancias al pais*/
        pef.setPCityDistance(formPreferencias);
        pef.setSelectedPCityDistance(sel);

        /*Insertamos en el formulario las distancias a la ciudad*/
        ArrayList formCityDistance = new ArrayList();
        //ArrayList cityDistance = new ArrayList(pef.getCitiesDistance());
        String[] cityDistance =pef.getCitiesDistance();
        for( int i=0; i<cityDistance.length; i++) {
            String pref = (String)cityDistance[i];
            formCityDistance.add(new Study(i+1,pref));
        }
        pef.setCityDistance(formCityDistance);
        pef.setSelectedCityDistance(sel);
        /*Insertamos en el formulario las Pdistancias a la ciudad*/
        pef.setPCountryDistance(formPreferencias);
        pef.setSelectedPCountryDistance(sel);
        return mapping.findForward("Insertar");
    }
}
```

1.2.6. es.ucm.fdi.gice4s.http.view.actionforms.studentout.**PreferenciasForm**

```
import javax.servlet.http.HttpServletRequest;
import org.apache.struts.action.ActionErrors;
import org.apache.struts.action.ActionMapping;
import es.ucm.fdi.gice4s.util.struts.action.DefaultActionForm;

public class PreferenciasForm extends DefaultActionForm
{

   /*Para idiomas*/
   private java.util.Collection languages;
    private int[] selectedLanguages;

    private java.util.Collection PLanguages;
    private int[] selectedPLanguages;

    /*Para paises*/
    private java.util.Collection countries;
    private int[] selectedCountries;

    private java.util.Collection PCountries;
    private int[] selectedPCountries;

    /*Para distancias a paises*/
    private java.util.Collection countryDistance;
    private int[] selectedCountryDistance;

    private java.util.Collection PCountryDistance;
    private int[] selectedPCountryDistance;

    /*Para distancias a ciudades*/
    private java.util.Collection cityDistance;
    private int[] selectedCityDistance;

    private java.util.Collection PCityDistance;
    private int[] selectedPCityDistance;

    /*Almacenamos las preferencias*/
    public static final String[] preferencias = {
            "No relevante",
            "Preferible",
            "Relevante",
            "Imprescindible"
    };
```

```
   /*Almacenamos las distancias a los paises*/
   public static final String [] countriesDistance = {
           "Cerca (<1000km)",
           "Lejos (1000km - 2000km)",
           "Muy Lejos (>2000km)"
   };

   /*Almacenamos las distancias a las ciudades*/
   public static final String [] citiesDistance = {
           "Menos de 10km",
           "Entre 10km y 20km",
           "Mas de 20km”  };

  public PreferenciasForm(){
       reset();
  }

  public void reset(ActionMapping mapping, HttpServletRequest request) {
       reset();
  }


  public void reset(){

        this.selectedLanguages = new int[]{};
       this.selectedCityDistance = new int[]{};
       this.selectedCountries = new int[]{};
       this.selectedCountryDistance = new int[]{};
       /*Las preferencias*/
       this.selectedPCityDistance = new int[]{};
       this.selectedPCountryDistance = new int[]{};
       this.selectedPLanguages = new int[]{};
       this.selectedPCountries = new int[]{};

  }

 public ActionErrors validate(ActionMapping mapping,
                  HttpServletRequest request) {
      ActionErrors errors = new ActionErrors();
      return errors;

 }

 public int[] getSelectedLanguages(){
       return selectedLanguages;
 }

 public void setSelectedLanguages(int[] selectedLanguages){
       this.selectedLanguages = selectedLanguages;
 }

 public java.util.Collection getLanguages(){
       return languages;
 }

public void setLanguages(java.util.Collection languages){
       this.languages = languages;
}

 public int[] getSelectedPLanguages() {
       return selectedPLanguages;
 }

 public void setSelectedPLanguages(int[] selectedLanguages) {
       this.selectedPLanguages = selectedLanguages;
 }

 public java.util.Collection getPLanguages() {
```

```
     return PLanguages;
}

public void setPLanguages(java.util.Collection languages) {
     this.PLanguages = languages;
}

public int[] getSelectedCountries() {
     return selectedCountries;
}

public void setSelectedCountries(int[] selectedCountries) {
     this.selectedCountries = selectedCountries;
}

public java.util.Collection getCountries() {
     return countries;
}

public void setCountries(java.util.Collection Countries) {
     this.countries = Countries;
}

public int[] getSelectedPCountries() {
     return selectedPCountries;
}

public void setSelectedPCountries(int[] selectedCountries) {
     this.selectedPCountries = selectedCountries;
}

public java.util.Collection getPCountries() {
     return PCountries;
}

public void setPCountries(java.util.Collection Countries) {
     this.PCountries = Countries;
}
 /*Metodos para cityDistance*/

public int[] getSelectedCityDistance() {
     return selectedCityDistance;
}

public void setSelectedCityDistance(int[] selectedCityDistance) {
     this.selectedCityDistance = selectedCityDistance;
}

public java.util.Collection getCityDistance() {
     return cityDistance;
}

public void setCityDistance(java.util.Collection CityDistance) {
     this.cityDistance = CityDistance;
}

public int[] getSelectedPCityDistance() {
     return selectedPCityDistance;
}

public void setSelectedPCityDistance(int[] selectedCityDistance) {
     this.selectedPCityDistance = selectedCityDistance;
}

public java.util.Collection getPCityDistance() {
     return PCityDistance;
}
```

```
   public void setPCityDistance(java.util.Collection CityDistance) {
        this.PCityDistance = CityDistance;
   }

    /*Metodos para countryDistance*/
   public int[] getSelectedCountryDistance() {
        return selectedCountryDistance;
   }

   public void setSelectedCountryDistance(int[] selectedCountryDistance) {
        this.selectedCountryDistance = selectedCountryDistance;
   }

   public java.util.Collection getCountryDistance() {
        return countryDistance;
   }

   public void setCountryDistance(java.util.Collection CountryDistance) {
        this.countryDistance = CountryDistance;
   }

   public int[] getSelectedPCountryDistance() {
        return selectedPCountryDistance;
   }

   public void setSelectedPCountryDistance(int[] selectedCountryDistance) {
        this.selectedPCountryDistance = selectedCountryDistance;
   }

   public java.util.Collection getPCountryDistance() {
        return PCountryDistance;
   }

   public void setPCountryDistance(java.util.Collection CountryDistance) {
        this.PCountryDistance = CountryDistance;
   }

   public String[] getPreferencias(){
        return preferencias;
   }

   public String[] getCountriesDistance() {
        return countriesDistance;
   }

   public String[] getCitiesDistance() {
        return citiesDistance;
   }

}
```

1.2.7. es.ucm.fdi.gice4s.http.controller.actions.studentout.**PreferenciasAction**

```
import java.io.IOException;
import java.util.ArrayList;
import java.util.LinkedList;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;
import org.apache.struts.action.ActionForm;
import org.apache.struts.action.ActionForward;
import org.apache.struts.action.ActionMapping;
import es.ucm.fdi.gice4s.http.controller.session.DataManager;
import es.ucm.fdi.gice4s.http.controller.session.SessionManager;
import es.ucm.fdi.gice4s.http.view.actionforms.studentout.PreferenciasForm;
import es.ucm.fdi.gice4s.model.Preferencias.Eleccion;
```

```
import es.ucm.fdi.gice4s.model.Preferencias.Inteligencia;
import es.ucm.fdi.gice4s.util.exceptions.InternalErrorException;
import es.ucm.fdi.gice4s.util.struts.action.DefaultAction;

public class PreferenciasAction extends DefaultAction
{


    protected ActionForward doExecute(ActionMapping mapping, ActionForm form,
                                HttpServletRequest request;
                                HttpServletResponse response) throws
                                IOException, ServletException,
                                InternalErrorException
    {
       LinkedList listaElecciones = new LinkedList();
       PreferenciasForm pef = (PreferenciasForm)form;
        /*Recogemos el idioma que ha seleccionado*/
        ArrayList languages = new ArrayList(DataManager.getLanguages());
        int[] res = pef.getSelectedLanguages();
        String language = ((String)languages.get(res[0]-1));
        /*Recogemos las preferencias*/
        String[] preferencias =pef.getPreferencias();
        res = pef.getSelectedPLanguages();
        String prefLanguage = ((String)preferencias[res[0]-1]);

        /*Recogemos el pais que ha seleccionado*/
        ArrayList countries = new ArrayList(DataManager.getCountries());
        res = pef.getSelectedCountries();
        String country = ((String)countries.get(res[0]-1));
        /*Recogemos las preferencias*/
        res = pef.getSelectedPCountries();
        String prefCountry = ((String)preferencias[res[0]-1]);

        /*Recogemos la distancia al pais que ha seleccionado*/
        String[] distanceP =pef.getCountriesDistance();
        res = pef.getSelectedCountryDistance();
        String countryDistance = ((String)distanceP[res[0]-1]);
        /*Recogemos las preferencias*/
        res = pef.getSelectedPCountryDistance();
        String prefCountryDistance = ((String)preferencias[res[0]-1]);

        /*Recogemos la distancia a la ciudad que ha seleccionado*/
        String[] distanceC = pef.getCitiesDistance();
        res = pef.getSelectedCityDistance();
        String cityDistance =  ((String)distanceC[res[0]-1]);
        /*Recogemos las preferencias*/
        res = pef.getSelectedPCityDistance();
        String prefCityDistance = ((String)preferencias[res[0]-1]);

        /*Creamos objetos con las elecciones introducidas y los añadimos
         * a la lista listaElecciones*/
        Eleccion eIdioma = new Eleccion("Language", language,
                devuelveImportancia(prefLanguage));
        listaElecciones.addLast(eIdioma);
        Eleccion ePais = new Eleccion("Country", country,
                devuelveImportancia(prefCountry));
        listaElecciones.addLast(ePais);
        Eleccion eDistanciaPais = new Eleccion("DistanceCountry",
                    countryDistance, devuelveImportancia(prefCountryDistance));
        listaElecciones.addLast(eDistanciaPais);
        Eleccion eDistanciaCiudad = new Eleccion("DistanceCity", cityDistance,
                devuelveImportancia(prefCityDistance));
        listaElecciones.addLast(eDistanciaCiudad);

        /*Busqueda en la base de datos de las universidades que coincidan con
         * las preferencias indicadas*/
        Inteligencia inteligencia = new Inteligencia(listaElecciones);
        LinkedList listaMejores = inteligencia.devuelveListaMejores();
```

```
        /*Guardamos en la sesion la lista de universidades seleccionadas*/
        HttpSession session = request.getSession();
        session.setAttribute(SessionManager.UNIVERSITIES_LIST,listaMejores);
        /*Para mostrar posteriormente las asignaturas*/
       return mapping.findForward("Preferencias");
    }

    int devuelveImportancia(String importancia) {
        if (importancia.equals("Imprescindible")) {
          return Eleccion.E_IMPRESCINDIBLE;
        }
        else if (importancia.equals("Importante")) {
          return Eleccion.E_IMPORTANTE;
        }
        else if (importancia.equals("Relevante")) {
          return Eleccion.E_RELEVANTE;
        }
        else if (importancia.equals("Preferible")) {
          return Eleccion.E_PREFERIBLE;
        }
        else if (importancia.equals("No relevante")) {
          return Eleccion.E_NO_RELEVANTE;
        }
        else {
          return -1;
        }
      }
}
```

1.2.8. es.ucm.fdi.gice4s.http.controller.actions.studentout.**RecogerAction**

```
import java.io.IOException;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;
import org.apache.struts.action.ActionForm;
import org.apache.struts.action.ActionForward;
import org.apache.struts.action.ActionMapping;
import es.ucm.fdi.gice4s.http.controller.session.DataManager;
import es.ucm.fdi.gice4s.http.controller.session.SessionManager;
import es.ucm.fdi.gice4s.util.exceptions.InternalErrorException;
import es.ucm.fdi.gice4s.util.struts.action.DefaultAction;
import java.util.*;

public class RecogerParamAction extends DefaultAction
{
    public static final String Faculty = "faculty";
    public static final String Universidad = "universidad";
    public static final String Convenios = "convenios";



    protected ActionForward doExecute(ActionMapping mapping, ActionForm form,
                                HttpServletRequest request,
                                HttpServletResponse response) throws
                                IOException, ServletException,
                                InternalErrorException
    {
       HttpSession session = request.getSession();
       String faculty  = (String)session.getAttribute(SessionManager.FACULTY);
       String  universidad =
                     (String)session.getAttribute(SessionManager.UNIVERSITY);
       if(faculty == null) {
              universidad = request.getParameter("universidad");
              ArrayList faculties =
                     (ArrayList)DataManager.getFaculties(universidad);
```

```
            faculty = (String)faculties.get(0);
        }
        session.setAttribute(SessionManager.FACULTY, faculty);
        request.setAttribute(Faculty , faculty);
        request.setAttribute(Universidad , universidad);
        /*Recogemos los nombres de las areas de conocimiento*/
        ArrayList convenios = (ArrayList)DataManager.getAgreements();
        ArrayList conveniosHW =(ArrayList)convenios.get(0);
        ArrayList conveniosSW =(ArrayList)convenios.get(1);
        ArrayList conveniosMT =(ArrayList)convenios.get(2);
        ArrayList conveniosPH =(ArrayList)convenios.get(3);
        ArrayList conveniosOT =(ArrayList)convenios.get(4);
        session.setAttribute(SessionManager.CONVENIOS_HW, conveniosHW);
        session.setAttribute(SessionManager.CONVENIOS_SW, conveniosSW);
        session.setAttribute(SessionManager.CONVENIOS_MT, conveniosMT);
        session.setAttribute(SessionManager.CONVENIOS_PH, conveniosPH);
        session.setAttribute(SessionManager.CONVENIOS_OT, conveniosOT);

        return mapping.findForward("Recoger");
    }
}
```

1.2.9.es.ucm.fdi.gice4s.http.view.actionforms.studentout.**SelectedSubjectForm**

```
import javax.servlet.http.HttpServletRequest;
import org.apache.struts.action.ActionErrors;
import org.apache.struts.action.ActionMapping;
import es.ucm.fdi.gice4s.util.struts.action.DefaultActionForm;

public class selectedSubjectForm extends DefaultActionForm
{
    private String[] selectedPendingSubject;
    private String[] selectedSubject;
    private String[] selectedSubjectProvisional;
    private String selectedUniversity;

    public selectedSubjectForm()
    {
        reset();
    }

    public void reset(ActionMapping mapping, HttpServletRequest request) {
        reset();
    }

    public void reset()
    {
        this.selectedPendingSubject = new String[]{};
        this.selectedSubject = new String []{};
        this.selectedUniversity = null;
        this.selectedSubjectProvisional= new String []{};

    }

    public ActionErrors validate(ActionMapping mapping,
                         HttpServletRequest request) {
       ActionErrors errors = new ActionErrors();
        return errors;
    }

    public String[] getSelectedSubject() {
        return selectedSubject;
    }


    public String[] getSelectedPendingSubject() {
        return selectedPendingSubject;
```

```
    }

    public String[] getSelectedSubjectProvisional() {
        return selectedSubjectProvisional;
    }

    public void setSelectedSubject(String[] selected) {
        this.selectedSubject = selected;
    }

    public void setSelectedPendingSubject(String[] selected) {
        this.selectedPendingSubject = selected;
    }

    public void setSelectedSubjectProvisional(String[] selected) {
        this.selectedSubjectProvisional = selected;
    }

    public void setSelectedUniversity(String selected) {
        this.selectedUniversity = selected;
    }

    public String getSelectedUniversity() {
        return this.selectedUniversity;
    }
}
```

1.2.10. es.ucm.fdi.gice4s.http.controller.action.studentout. **MostrarPreferenciasAction**

```
import java.io.IOException;
import java.util.*;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;
import org.apache.struts.action.ActionForm;
import org.apache.struts.action.ActionForward;
import org.apache.struts.action.ActionMapping;
import es.ucm.fdi.gice4s.http.controller.session.DataManager;
import es.ucm.fdi.gice4s.http.controller.session.SessionManager;
import es.ucm.fdi.gice4s.http.view.actionforms.studentout.selectedSubjectForm;
import es.ucm.fdi.gice4s.util.exceptions.InternalErrorException;
import es.ucm.fdi.gice4s.util.struts.action.DefaultAction;
import es.ucm.fdi.gice4s.model.Preferencias.EntradaBD;
import es.ucm.fdi.gice4s.model.Similitud.Asignatura;
import es.ucm.fdi.gice4s.model.Similitud.SimilitudAsignaturas;


public class MostrarPreferenciasAction extends DefaultAction {

  public static final String Universidades = "resultado";
  LinkedList resultado;

  protected ActionForward doExecute(ActionMapping mapping, ActionForm form,
       HttpServletRequest request,
       HttpServletResponse response) throws IOException, ServletException,
       InternalErrorException {
    resultado = new LinkedList();
    /*Recogemos los valores de las universidaes seleccionadas*/
    HttpSession session = request.getSession();
    LinkedList listaMejores  =
       (LinkedList)session.getAttribute(SessionManager.UNIVERSITIES_LIST);

    /*Recogemos los valores de las asignaturas que ha introducido*/
    selectedSubjectForm ssf = (selectedSubjectForm)form;
    String[] subjectPending = ssf.getSelectedPendingSubject();
```

```
Calendar rightNow = Calendar.getInstance();
int yActual = rightNow.get(Calendar.YEAR);
int yAnterior = yActual-1;
String curso=String.valueOf(yAnterior)+"/"+String.valueOf(yActual);

/*Con esto abrimos el fichero de la ontologia*/
SimilitudAsignaturas simil = new SimilitudAsignaturas();
if(subjectPending.length != 0){
  /*Recorremos la lista de universidades*/
  for(int i = 0; i<listaMejores.size(); i++){
    EntradaBD entrada = (EntradaBD)listaMejores.get(i);
    String name = entrada.getName();
    ArrayList facultades = (ArrayList)DataManager.getFaculties(name);
    String facultad = (String)facultades.get(0);
    /*Obtenemos las asignaturas de la universidad y las recorremos*/
    ArrayList asig = obtenerAsignaturas(curso,name);
    double maxSim = 0;
    double puntuacion = 0;
    for(int k = 0; k <subjectPending.length; k++){
      LinkedList conveniosK =
         DataManager.getConveniosAsignatura(subjectPending[k],"Facultad de
         Informática");
      String creditosK =
         DataManager.returnCredits(curso,subjectPending[k],"Facultad de
         Informática");
      float creditK = Float.parseFloat(creditosK);
      Asignatura asigK = new
         Asignatura(subjectPending[k],creditK,conveniosK);

      /*Comparamos la similitud entre cada asignatura de la universidad
      * destino y las que ha introducido el usuario*/
      double similitud = 0;
      for(int j = 0; j<asig.size(); j++){
        try{
          LinkedList conveniosJ =
                DataManager.getConveniosAsignatura((String)asig.get(j),
                facultad);
          String creditosJ =
                DataManager.returnCredits(curso,(String)asig.get(j),
                facultad);
          float creditJ = Float.parseFloat(creditosJ);
          Asignatura asigJ = new
                Asignatura((String)asig.get(j),creditJ,conveniosJ);
          similitud = simil.similitudAsignatura(asigK,asigJ);
          if(similitud>maxSim)
            maxSim = similitud;
        } catch (Exception e){
          System.out.println(e.getMessage());
        }
      }
      /*Comparamos con las asignaturas que ha introducido el usuario en el
         sistema*/
      LinkedList asigProvisional =
         DataManager.getProvisionalSubjects(facultad);
      if(asigProvisional.size() != 0) {
        for(int j = 0; j<asigProvisional.size(); j++){
          try {
            String creditosP =
                DataManager.returnCreditsProvisional(curso,
                (String)asigProvisional.get(j),facultad);
            float creditP = Float.parseFloat(creditosP);
            String enlace =
                DataManager.recuperarLink(curso,
                (String)asigProvisional.get(j),facultad);
            String convenios =
                DataManager.recuperarConvenios((String)asigProvisional.get(
                j), creditosP, enlace );
            LinkedList conveniosP = new LinkedList();
```

```
              /*Recorremos la lista de convenios y los vamos almacenando en
                  una lista*/
              StringTokenizer st = new StringTokenizer(convenios);
              while(st.hasMoreElements()) {
                String conv = st.nextToken();
                if(!conv.equals(";"))
                  conveniosP.addLast(conv);
              }
              Asignatura asigP = new
                  Asignatura((String)asig.get(j),creditP,conveniosP);
              similitud = simil.similitudAsignatura(asigK,asigP);
              if(similitud>maxSim)
                maxSim = similitud;
            } catch (Exception e){
              System.out.println(e.getMessage());
            }
          }
        }
        /*Vamos sumando la puntuacion de todas las asignaturas*/
        puntuacion = puntuacion + maxSim;
      }// fin del for(K)
      /*Calculamos la media de la universidad tratada y cambiamos su
          puntuación*/
      puntuacion = puntuacion / subjectPending.length;
      double puntuacionAnt = entrada.getPuntuacion();
      double puntuacionFinal = 0.0;
      if(puntuacionAnt == -1)
        puntuacionFinal = puntuacion;
      else {
        puntuacionFinal = (puntuacion * 0.5)+(puntuacionAnt * 0.5);
        puntuacionAnt = (int)(puntuacionAnt * 100);
      }
      puntuacionFinal = (int)(puntuacionFinal * 100);
      puntuacion = (int)(puntuacion * 100);
      entrada.setPuntuacion(puntuacionFinal);
      entrada.setPuntuacionAsig(puntuacion);
      entrada.setPuntuacionPref(puntuacionAnt);
      insertaOrden(puntuacionFinal, entrada);
    }
    /*Añadimos como atributo las universidades con su puntuacion*/

    request.setAttribute(Universidades , resultado);
  }
  /*Si no ha introducido ninguna asignatura que quiera cursar le muestra la
  * lista de universidaes obtenidas*/
  else{
    resultado = listaMejores;
    for(int i = 0; i < resultado.size(); i++) {
      EntradaBD entrada = (EntradaBD)resultado.get(i);
      double puntuacion = entrada.getPuntuacion();
      entrada.setPuntuacion((int)(puntuacion * 100));
      entrada.setPuntuacionAsig(-1);
      if (puntuacion != -1)
        entrada.setPuntuacionPref((int)(puntuacion * 100));
      else
        entrada.setPuntuacionPref(puntuacion);
      resultado.set(i,entrada);
    }
    request.setAttribute(Universidades , listaMejores);
  }
  return mapping.findForward("Mostrar");
}

private void insertaOrden(double puntuacion, EntradaBD entrada) {
  int i = 0;
  boolean encontrado = false;
  while ( (i < resultado.size()) && !encontrado) {
    if (puntuacion > ( (EntradaBD) resultado.get(i)).getPuntuacion()) {
```

```
        resultado.add(i, entrada);
        encontrado = true;
      }
      i++;
    }
    if(!encontrado)
      resultado.addLast(entrada);
  }


  private static ArrayList obtenerAsignaturas(String curso,String name) throws
            IOException, ServletException, InternalErrorException{
    ArrayList facultades = (ArrayList)DataManager.getFaculties(name);
    ArrayList resultado = new ArrayList();
    String facul = (String)facultades.get(0);
    ArrayList resultadoAux = (ArrayList)DataManager.getSubjects(curso,facul);
    resultado = (ArrayList)resultadoAux.get(0);
    return resultado;
  }
}
```

1.3. Apartado V.D. Sistema de ayuda al alumno para la elección de asignaturas a cursar.

A demás de las clases indicadas a continuación, en este apartado se emplean otras que ya han sido citadas anteriormente.

1.3.1. es.ucm.fdi.gice4s.http.controller.actions.studentout.**FindAllUniversitiesAction**

```
import java.io.IOException;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import org.apache.struts.action.ActionForm;
import org.apache.struts.action.ActionForward;
import org.apache.struts.action.ActionMapping;
import es.ucm.fdi.gice4s.http.controller.session.DataManager;
import es.ucm.fdi.gice4s.util.exceptions.InternalErrorException;
import es.ucm.fdi.gice4s.util.struts.action.DefaultAction;

public class FindAllUniversitiesAction extends DefaultAction
{

    public static final String FindAllUniversities = "universities";

    protected ActionForward doExecute(ActionMapping mapping, ActionForm form,
                                      HttpServletRequest request,
                                      HttpServletResponse response) throws
                                      IOException, ServletException,
                                      InternalErrorException
    {
       /* Tomamos titulaciones de la facultad del alumno*/
       java.util.Collection universities = DataManager.getUniversities();
       request.setAttribute(FindAllUniversities , universities);
        return mapping.findForward("ListadoUniversities");
    }
}
```

1.3.2. es.ucm.fdi.gice4s.http.controller.actions.studentout. **FindAllFacultiesAction**

```
import java.io.IOException;
import javax.servlet.ServletException;
```

```
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import org.apache.struts.action.ActionForm;
import org.apache.struts.action.ActionForward;
import org.apache.struts.action.ActionMapping;
import es.ucm.fdi.gice4s.http.controller.session.DataManager;
import es.ucm.fdi.gice4s.util.exceptions.InternalErrorException;
import es.ucm.fdi.gice4s.util.struts.action.DefaultAction;

public class FindAllFacultiesAction extends DefaultAction
{

    public static final String FindAllFaculties = "faculties";
    public static final String UniversityName = "universidad";


    protected ActionForward doExecute(ActionMapping mapping, ActionForm form,
                                HttpServletRequest request,
                                HttpServletResponse response) throws
                                IOException, ServletException,
                                InternalErrorException
    {
       /* Tomamos titulaciones de la facultad del alumno*/
       String universidad = (String)request.getParameter("universidad");
       java.util.Collection faculties = DataManager.getFaculties(universidad);
       request.setAttribute(FindAllFaculties , faculties);
       request.setAttribute(UniversityName, universidad);
        return mapping.findForward("ListadoFaculties");
    }
}
```

1.3.3. es.ucm.fdi.gice4s.http.controller.actions.studentout. **MostrarAsignaturasAction**

```
import java.io.IOException;
import java.util.ArrayList;
import java.util.Calendar;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;
import org.apache.struts.action.ActionForm;
import org.apache.struts.action.ActionForward;
import org.apache.struts.action.ActionMapping;
import es.ucm.fdi.gice4s.http.controller.session.DataManager;
import es.ucm.fdi.gice4s.http.controller.session.SessionManager;
import es.ucm.fdi.gice4s.model.clasesAux.*;
import es.ucm.fdi.gice4s.model.userprofile.vo.*;
import es.ucm.fdi.gice4s.util.exceptions.InternalErrorException;
import es.ucm.fdi.gice4s.util.struts.action.DefaultAction;

public class MostrarAsignaturasAction extends DefaultAction
{
    public static final String FindAllSubjects = "mostrar";
    public static final String FindAllSubjects2 = "mostrarProvisional";
    public static final String FindAllPendingSubjects = "pendingSubjects";
    public static final String Faculty = "faculty";
    public static final String Universidad = "universidad";
    public static final String Convenios = "convenios";


    protected ActionForward doExecute(ActionMapping mapping, ActionForm form,
                                      HttpServletRequest request,
                                      HttpServletResponse response) throws
                                      IOException, ServletException,
                                      InternalErrorException
    {
```

```
HttpSession session = request.getSession();
String universidad = request.getParameter("universidad");
session.setAttribute(SessionManager.UNIVERSITY, universidad);
String asistente = request.getParameter("asistente");
session.setAttribute(SessionManager.ASISTENTE, asistente);
request.setAttribute(Universidad , universidad);

/*Recogemos los nombres de las areas de conocimiento*/
ArrayList convenios = (ArrayList)DataManager.getAgreements();
ArrayList conveniosHW =(ArrayList)convenios.get(0);
ArrayList conveniosSW =(ArrayList)convenios.get(1);
ArrayList conveniosMT =(ArrayList)convenios.get(2);
ArrayList conveniosPH =(ArrayList)convenios.get(3);
ArrayList conveniosOT =(ArrayList)convenios.get(4);
session.setAttribute(SessionManager.CONVENIOS_HW, conveniosHW);
session.setAttribute(SessionManager.CONVENIOS_SW, conveniosSW);
session.setAttribute(SessionManager.CONVENIOS_MT, conveniosMT);
session.setAttribute(SessionManager.CONVENIOS_PH, conveniosPH);
session.setAttribute(SessionManager.CONVENIOS_OT, conveniosOT);
Calendar rightNow = Calendar.getInstance();
 int yActual = rightNow.get(Calendar.YEAR);
 int yAnterior = yActual-1;
 String curso=String.valueOf(yAnterior)+"/"+String.valueOf(yActual);
String faculty = request.getParameter("facultad");
session.setAttribute(SessionManager.FACULTY,faculty);
request.setAttribute(Faculty , faculty);
//Tomamos las asignaturas de la facultad seleccionada
ArrayList subjects = (ArrayList)DataManager.getSubjects(curso,faculty);
ArrayList name = (ArrayList)subjects.get(0);
ArrayList enlace = (ArrayList)subjects.get(1);
ArrayList provisionalName = (ArrayList)subjects.get(2);
//ArrayList provisionalEnlace = (ArrayList)subjects.get(3);
ArrayList mostrar = new ArrayList();
ArrayList mostrarProvisional = new ArrayList();
/*Recogemos las asignaturas no introducidas por el alumno*/
for (int i = 0; i< name.size(); i++){
      String a = (String)name.get(i);
      String b = (String)enlace.get(i);
      //Comprobamos que los enlaces no sean vacios y añadimos a mostrar
      if (b != null){
            SubjectsLinks c = new SubjectsLinks (a, b);
            mostrar.add(c);
            }
      else{
            SubjectsLinks c = new SubjectsLinks (a,"");
            mostrar.add(c);
      }
}//end for
/*Recogemos las asignaturas introducidas por el alumno*/
      for (int i = 0; i< provisionalName.size(); i++){
            String a = (String)provisionalName.get(i);
            String b = DataManager.recuperarLink(curso,a,faculty);
            //Comprobamos que los enlaces no sean vacios y añadimos a
            mostrar
            if (b != null){
                  SubjectsLinks c = new SubjectsLinks (a, b);
                  mostrarProvisional.add(c);
                  }
            else{
                  SubjectsLinks c = new SubjectsLinks (a,"");
                  mostrarProvisional.add(c);
            }
      }
request.setAttribute(FindAllSubjects , mostrar);
request.setAttribute(FindAllSubjects2 , mostrarProvisional);
//Tomamos las asignaturas de los últimos cursos del alumno
UserProfileVO user =SessionManager.findUserProfile(request);
UserProfileDetailsVO details = user.getUserProfileDetailsVO();
```

```
      String dni = details.getDni();
      java.util.Collection pendingSubjects =
                  DataManager.getPendingSubjects(dni);
      request.setAttribute(FindAllPendingSubjects , pendingSubjects);
      return mapping.findForward("Mostrar");
    }
}
```

1.3.4 es.ucm.fdi.gice4s.http.controller.action.studentout. **MostrarSugerenciasAction**

```
import java.io.IOException;
import java.util.*;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;
import org.apache.struts.action.ActionForm;
import org.apache.struts.action.ActionForward;
import org.apache.struts.action.ActionMapping;
import es.ucm.fdi.gice4s.http.controller.session.DataManager;
import es.ucm.fdi.gice4s.http.controller.session.SessionManager;
import es.ucm.fdi.gice4s.http.view.actionforms.studentout.selectedSubjectForm;
import es.ucm.fdi.gice4s.model.Similitud.Asignatura;
import es.ucm.fdi.gice4s.model.Similitud.SimilitudAsignaturas;
import es.ucm.fdi.gice4s.model.userprofile.vo.UserProfileDetailsVO;
import es.ucm.fdi.gice4s.model.userprofile.vo.UserProfileVO;
import es.ucm.fdi.gice4s.util.exceptions.InternalErrorException;
import es.ucm.fdi.gice4s.util.struts.action.DefaultAction;


public class MostrarSugerenciasAction extends DefaultAction {

  public static final String convalidacionTotalString = "convalidacionTotal";
  public static final String convalidacionParcialString =
      "convalidacionParcial";
  public static final String convalidacionInsuficienteString =
  "convalidacionInsuficiente";
  public static final String nombreAsigSeleccionada = "nombreAsig";
  public final static double UMBRAL_CONVALIDACION_OB = 0.8;
  public final static double UMBRAL_CONVALIDACION_OP = 0.6;
  public static double umbral = UMBRAL_CONVALIDACION_OB;

  LinkedList convalidacionTotal;
  LinkedList convalidacionNoTotal;
  LinkedList convalidacionParcial;


  protected ActionForward doExecute(ActionMapping mapping, ActionForm form,
  HttpServletRequest request,
  HttpServletResponse response) throws IOException, ServletException,
  InternalErrorException
  {
    convalidacionTotal = new LinkedList();
    convalidacionParcial = new LinkedList();
    convalidacionNoTotal = new LinkedList();
    UserProfileVO user =SessionManager.findUserProfile(request);
    UserProfileDetailsVO details = user.getUserProfileDetailsVO();
    String dni = details.getDni();
    ArrayList convalidaciones =
     (ArrayList)((ArrayList)DataManager.recuperarConvalidaciones(dni)).get(1);
    ArrayList asignaturasConv = obtenerAsignaturasConv(convalidaciones);

    /*Recogemos la facultad elegida*/
    HttpSession session = request.getSession();
    String facultad  = (String)session.getAttribute(SessionManager.FACULTY);
```

```
      String universidad  =
      (String)session.getAttribute(SessionManager.UNIVERSITY);
    Calendar rightNow = Calendar.getInstance();
    int yActual = rightNow.get(Calendar.YEAR);
    int yAnterior = yActual-1;
    String curso=String.valueOf(yAnterior)+"/"+String.valueOf(yActual);

    /*Recogemos los valores de las asignaturas que ha introducido*/
    selectedSubjectForm ssf = (selectedSubjectForm)form;
    String[] subjectPending = ssf.getSelectedPendingSubject();
    String caracter = DataManager.obtenerCaracter(curso,subjectPending[0]);
    if(caracter.equalsIgnoreCase("TR") || caracter.equalsIgnoreCase("OB"))
      umbral = UMBRAL_CONVALIDACION_OB;
    else if (caracter.equalsIgnoreCase("OP"))
      umbral = UMBRAL_CONVALIDACION_OP;
    else
      System.out.println("WARNING: Valor inesperado en la base de datos.
      Encontrado: Caracter = " + caracter);
    /*Con esto abrimos el fichero de la ontologia*/
    SimilitudAsignaturas simil = new SimilitudAsignaturas();
    ArrayList asig = obtenerAsignaturas(curso,universidad);
    double maxSim = 0;
    double puntuacion = 0;
   LinkedList conveniosK =
      DataManager.getConveniosAsignatura(subjectPending[0],"Facultad de
      Informática");
    String creditosK =
      DataManager.returnCredits(curso,subjectPending[0],"Facultad de
      Informática");
    float creditK = Float.parseFloat(creditosK);
    Asignatura asigK = new Asignatura(subjectPending[0],creditK,conveniosK);
    /*Comparamos la similitud entre cada asignatura de la universidad
    * destino y las que ha introducido el usuario*/
    double similitud = 0;
    for(int j = 0; j<asig.size(); j++){
      try{
        LinkedList conveniosJ =
             DataManager.getConveniosAsignatura((String)asig.get(j),facultad);
        String creditosJ =
             DataManager.returnCredits(curso,(String)asig.get(j),facultad);
        float creditJ = Float.parseFloat(creditosJ);
        Asignatura asigJ = new
             Asignatura((String)asig.get(j),creditJ,conveniosJ);
        if(convalidaciones.contains(asigJ.getNombre()))
          asigJ.setConvalidada(1);
        else
          asigJ.setConvalidada(0);
        similitud = simil.similitudAsignatura(asigK,asigJ);
        asigJ.similitud = (int)(similitud * 100.0);
        if(similitud > umbral)
          convalidacionTotal.addLast(asigJ);
        else
          insertaOrden(asigJ, convalidacionNoTotal);
      } catch (Exception e){
        System.out.println(e.getMessage());
      }
    }
    dividirConvalidaciones(asigK);
    request.setAttribute(convalidacionTotalString, convalidacionTotal);
    request.setAttribute(convalidacionParcialString, convalidacionParcial);
    request.setAttribute(convalidacionInsuficienteString,
      convalidacionNoTotal);
    request.setAttribute(nombreAsigSeleccionada, asigK.getNombre());
    return mapping.findForward("Sugerir");
  }

  private void dividirConvalidaciones(Asignatura asigSeleccionada) {
    SimilitudAsignaturas simil = new SimilitudAsignaturas();
```

```
    for(int i = 0; i < convalidacionNoTotal.size(); i++) {
      for(int j = i; j < convalidacionNoTotal.size(); j++) {
        Asignatura[] asignaturasCombinadas = new Asignatura[2];
        asignaturasCombinadas[0] = (Asignatura)convalidacionNoTotal.get(i);
        asignaturasCombinadas[1] = (Asignatura)convalidacionNoTotal.get(j);
        Asignatura[] asignaturaSelec = new Asignatura[1];
        asignaturaSelec[0] = asigSeleccionada;
        double similitud = simil.similitudAsignatura(asignaturaSelec,
             asignaturasCombinadas);
        if(similitud > umbral) {
          Asignatura asignatura1 = new
             Asignatura((Asignatura)convalidacionNoTotal.get(i));
          Asignatura asignatura2 = new
             Asignatura((Asignatura)convalidacionNoTotal.get(j));
          similitud = (int)(similitud * 100.0);
          asignatura1.similitud = similitud;
          asignatura2.similitud = similitud;
          convalidacionParcial.addLast(asignatura1);
          convalidacionParcial.addLast(asignatura2);
        }
      }
    }
  }

  private void insertaOrden(Asignatura asig, LinkedList lista) {
    int i = 0;
    boolean encontrado = false;
    while ( (i < lista.size()) && !encontrado) {
      if (asig.similitud > ((Asignatura)lista.get(i)).similitud) {
        lista.add(i, asig);
        encontrado = true;
      }
      i++;
    }
    if(!encontrado)
      lista.addLast(asig);
  }


  private static ArrayList obtenerAsignaturas(String curso,String name)throws
       IOException, ServletException, InternalErrorException{
    ArrayList facultades = (ArrayList)DataManager.getFaculties(name);
    ArrayList resultado = new ArrayList();
    String facul = (String)facultades.get(0);
    ArrayList resultadoAux = (ArrayList)DataManager.getSubjects(curso,facul);
    resultado = (ArrayList)resultadoAux.get(0);
    return resultado;
  }

  private ArrayList obtenerAsignaturasConv(ArrayList convalidaciones) {
    ArrayList asignaturasConv = new ArrayList();
    for(int i = 0; i< convalidaciones.size(); i++) {
      String listaAsig = (String)convalidaciones.get(i);
      StringTokenizer tk = new StringTokenizer(listaAsig, ";");
      while(tk.hasMoreTokens()) {
        String asig = tk.nextToken();
        asignaturasConv.add(asig);
      }
    }
    return asignaturasConv;
  }
}
```

1.3.5. es.ucm.fdi.gice4s.http.controller.actions.studentout. **MostrarConvalidacionesAction**

```
import java.io.IOException;
import java.util.ArrayList;
import java.util.Calendar;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;
import org.apache.struts.action.ActionForm;
import org.apache.struts.action.ActionForward;
import org.apache.struts.action.ActionMapping;
import es.ucm.fdi.gice4s.http.controller.session.DataManager;
import es.ucm.fdi.gice4s.http.controller.session.SessionManager;
import es.ucm.fdi.gice4s.model.userprofile.vo.*;
import es.ucm.fdi.gice4s.model.clasesAux.*;
import es.ucm.fdi.gice4s.util.exceptions.InternalErrorException;
import es.ucm.fdi.gice4s.util.struts.action.DefaultAction;

public class MostrarConvalidacionesAction extends DefaultAction
{
    public static final String FindAllSubjects = "elegidas";
    public static final String FindAllPendingSubjects = "pendientes";
    public static final String Faculty = "faculty";
    public static final String Universidad = "universidad";
    public static final String Name = "name";
    public static final String SurName = "surName";
    public static final String DNI = "dni";
    public static final String Convalidaciones = "convalidaciones";
    public static final String creditosTotalesAqui = "creditosTotalesAqui";
    public static final String creditosTotalesAlli = "creditosTotalesAlli";
    public static final String libreElec = "libreElec";
    public static final String Asistente = "0";
    public static final String Curso = "curso";


    protected ActionForward doExecute(ActionMapping mapping, ActionForm form,
                                      HttpServletRequest request,
                                      HttpServletResponse response) throws
                                      IOException, ServletException,
                                      InternalErrorException
    {
       UserProfileVO user =SessionManager.findUserProfile(request);
       UserProfileDetailsVO details = user.getUserProfileDetailsVO();
       String name = details.getFirstName();
       String surName = details.getSurname();
       String dni = details.getDni();
       request.setAttribute(Name , name);
       request.setAttribute(SurName, surName);
       request.setAttribute(DNI , dni);
       Calendar rightNow = Calendar.getInstance();
        int yActual = rightNow.get(Calendar.YEAR);
        int yAnterior = yActual-1;
        String curso=String.valueOf(yAnterior)+"/"+String.valueOf(yActual);
       /*Devuelve el array que contiene las convalidaciones*/
       ArrayList subjects =
             (ArrayList)DataManager.recuperarConvalidaciones(curso,dni);
        /*Tomamos del array las pendientes y las elegidas*/
       ArrayList pendientes = (ArrayList)subjects.get(0);
       ArrayList elegidas = (ArrayList)subjects.get(1);
       ArrayList creditos = (ArrayList)subjects.get(2);
       ArrayList creditosT = (ArrayList)subjects.get(3);
       ArrayList creditosAlli = (ArrayList)subjects.get(4);
       ArrayList totalAlli = (ArrayList)subjects.get(5);
       float totalAqui= 0;
       float totalesAlli= 0;
```

```
		ArrayList convalidaciones = new ArrayList();
		for (int i = 0; i< pendientes.size(); i++){
			String a = (String)pendientes.get(i);
			String b = (String)elegidas.get(i);
			String c = (String)creditos.get(i);
			String d = (String)creditosT.get(i);
			totalAqui = totalAqui + Float.parseFloat(d);
			String f = (String)creditosAlli.get(i);
			String g = (String)totalAlli.get(i);
			totalesAlli = totalesAlli + Float.parseFloat(g);
			Convalidacion e = new Convalidacion (b, a,c,f);
			convalidaciones.add(e);
		}
		request.setAttribute(Convalidaciones , convalidaciones);
		HttpSession session = request.getSession();
		String faculty  = (String)session.getAttribute(SessionManager.FACULTY);
		String asistente =
				(String)session.getAttribute(SessionManager.ASISTENTE);
		System.out.println(asistente);
		String universidad  =
				(String)session.getAttribute(SessionManager.UNIVERSITY);
		String creditoAqui = String.valueOf(totalAqui);
		request.setAttribute(creditosTotalesAqui , creditoAqui);
		String creditoAlli = String.valueOf(totalesAlli);
		request.setAttribute(creditosTotalesAlli , creditoAlli);
		request.setAttribute(Universidad , universidad);
		request.setAttribute(Faculty , faculty);
		request.setAttribute(Asistente, asistente);
		request.setAttribute(Curso, curso);
		/* La correspondencia entre creditos aqui y alli es:
		70 ECTS = 60 aqui--> hacemos
		 * una regla de tres para calcular cuantos creditos le sobran y
		convalidarselos como libre eleccion
		 */
		float x = (70*totalesAlli)/60;
		float libreEleccion = x - totalAqui;
		libreEleccion = (int)(libreEleccion * 100)/100;
		if (libreEleccion < 0){
			String libre = String.valueOf(0);
			request.setAttribute(libreElec ,libre);
		}
		else{
			String libre = String.valueOf(libreEleccion);
			request.setAttribute(libreElec , libre);
		}
		return mapping.findForward("Mostrar");
	}

}
```

1.4 Apartado V.E. Sistema de ayuda al coordinador Erasmus para decidir convalidaciones

A demás de las clases indicadas a continuación, en este apartado se emplean otras que ya han sido citadas anteriormente.

1.4.1. es.ucm.fdi.gice4s.http.controller.actions.studentout. **MostrarResultadosConvalidacionAction**

```
import java.io.IOException;
import java.util.*;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
```

```
import javax.servlet.http.HttpSession;
import org.apache.struts.action.ActionForm;
import org.apache.struts.action.ActionForward;
import org.apache.struts.action.ActionMapping;
import es.ucm.fdi.gice4s.http.controller.session.DataManager;
import es.ucm.fdi.gice4s.http.controller.session.SessionManager;
import es.ucm.fdi.gice4s.http.view.actionforms.studentout.selectedSubjectForm;
import es.ucm.fdi.gice4s.model.Similitud.Asignatura;
import es.ucm.fdi.gice4s.model.Similitud.SimilitudAsignaturas;
import es.ucm.fdi.gice4s.util.exceptions.InternalErrorException;
import es.ucm.fdi.gice4s.util.struts.action.DefaultAction;

public class MostrarResultadosConvalidacionAction extends DefaultAction
{

        public static final String Similitud = "similitud";
        public static final String Umbral = "umbral";
        public static final String ListaAqui = "listaAqui";
        public static final String ListaAlli = "listaAlli";
        public static final String Resultado = "resultado";
        public final static double UMBRAL_CONVALIDACION_OB = 0.8;
        public final static double UMBRAL_CONVALIDACION_OP = 0.6;
        public static double umbral = UMBRAL_CONVALIDACION_OB;

    protected ActionForward doExecute(ActionMapping mapping, ActionForm
                                      form, HttpServletRequest request,
                                      HttpServletResponse response) throws
                                      IOException, ServletException,
                                      InternalErrorException
    {
       /*Recogemos los valores de las asignaturas que ha introducido*/
       selectedSubjectForm ssf = (selectedSubjectForm)form;
       String[] subjectPending = ssf.getSelectedPendingSubject();
       String[] selectedSubject = ssf.getSelectedSubject();

       Calendar rightNow = Calendar.getInstance();
        int yActual = rightNow.get(Calendar.YEAR);
        int yAnterior = yActual-1;
        String curso=String.valueOf(yAnterior)+"/"+String.valueOf(yActual);
       /*Recogemos la facultad elegida*/
       HttpSession session = request.getSession();
       String facultad  = (String)session.getAttribute(SessionManager.FACULTY);
       Asignatura[] asignaturasAqui = new Asignatura[subjectPending.length];
       for(int i = 0; i< subjectPending.length; i++) {
              LinkedList convenios =
              DataManager.getConveniosAsignatura(subjectPending[i],"Facultad de
              Informática");
              String creditos =
              DataManager.returnCredits(curso,subjectPending[i],"Facultad de
              Informática");
              float credit = Float.parseFloat(creditos);
              Asignatura asig = new Asignatura(subjectPending[i], credit,
                     convenios);
              asignaturasAqui[i] = asig;
       }
       Asignatura[] asignaturasAlli = new Asignatura[selectedSubject.length];
       for(int i = 0; i< selectedSubject.length; i++) {
              LinkedList convenios =
              DataManager.getConveniosAsignatura(selectedSubject[i], facultad);
              String creditos =
              DataManager.returnCredits(curso,selectedSubject[i], facultad);
              float credit = Float.parseFloat(creditos);
              Asignatura asig = new Asignatura(selectedSubject[i], credit,
                     convenios);
              asignaturasAlli[i] = asig;
       }
       SimilitudAsignaturas simil = new SimilitudAsignaturas();
       LinkedList listaAqui = new LinkedList();
```

```
        LinkedList listaAlli = new LinkedList();
        for(int i = 0; i < asignaturasAqui.length; i++)
              listaAqui.addLast(asignaturasAqui[i]);
        for(int i = 0; i < asignaturasAlli.length; i++)
              listaAlli.addLast(asignaturasAlli[i]);
        double similitud = simil.similitudAsignatura(asignaturasAqui,
        asignaturasAlli);
        similitud = Math.round(similitud*100)/100.0;
        request.setAttribute(Similitud, new Double (similitud));
        request.setAttribute(Umbral, new Double(umbral));
        request.setAttribute(ListaAqui, listaAqui);
        request.setAttribute(ListaAlli, listaAlli);
        String resultado = "invalido";
        if (similitud >= umbral)
              resultado = "valido";
        request.setAttribute(Resultado,resultado);
        return mapping.findForward("Mostrar");
    }
}
```

2. RECUPERADOR DE INFORMACIÓN

Las principales clases de esta aplicación, que se explicó en el apartado A del capítulo V, son las siguientes:

2.1. ri.Entrenador

```
package ri;

import java.util.*;
import java.io.*;
import javax.swing.*;

/**
 * <p>Title: Recuperador de Información</p>
 *
 * <p>Description: Recuperador de Información</p>
 *
 * <p>Copyright: Copyright (c) 2005</p>
 *
 * <p>Company: </p>
 *
 * @author M. Fuertes, P. Gallo, N. Ortiz de Zárate
 * @version 1.0
 */
public class Entrenador {
  /**
   * Lista de las frecuencias de las palabras
   */
  public LinkedList listasPalabras = new LinkedList();
  /**
   * Lista de pesos
   */
  public LinkedList listasPesos = new LinkedList();
  /**
   * Lista de categorías
   */
  public LinkedList categorias = new LinkedList();
  /**
   * Lista de listas de textos (una por categoría)
   */
  public LinkedList textos = new LinkedList();
  private String ruta = System.getProperty("user.dir");
  private JFileChooser sfc = new JFileChooser(ruta);

  /**
   * Constructora de la clase
   */
  public Entrenador() {
    super();
  }

  /**
   * Método que realiza el entrenamiento. Genera vectores para cada categoría
   * a partir de los textos.
   * @param cat LinkedList Lista (de String) de los nombres de las categorías
   * @param tex LinkedList Lista (de LinkedList de String) de los textos de
   * cada categoría
   */
  public void entrenar(LinkedList cat, LinkedList tex) {
    categorias = cat;
    textos = tex;
    inicializarListas(categorias.size());
    for(int i = 0; i < categorias.size(); i++) {
      Transformador transformador = new Transformador();
```

```
    LinkedList textosCategoria = ((LinkedList)textos.get(i));
    for(int j = 0; j < textosCategoria.size(); j++) {
      String texto = (String)textosCategoria.get(j);
      LinkedList palabrasTexto = transformador.analizar(texto);
      anadirPalabras(palabrasTexto, i, j);
    }
  }
  generarPesos();
  guardarResultados();
}

/**
 * Método que inicializa las listas
 * @param tamano int Tamaño que tendrán las listas
 */
private void inicializarListas(int tamano) {
  for(int i = 0; i < tamano; i++) {
    listasPalabras.addLast(new LinkedList());
    //listasPesos.addLast(new LinkedList());
  }
}

/**
 * Método que añade las palabras a la lista de palabras
 * @param lista LinkedList Lista (de Elemento) que contiene las palabras que
 * vamos a añadir
 * @param categoria int Número de categoría a la que corresponden las
 * @param texto int Número de texto dentro de una categoría a la que
 * corresponden las palabras
 */
private void anadirPalabras(LinkedList lista, int categoria, int texto) {
  LinkedList palabras = ((LinkedList)listasPalabras.get(categoria));
  for(int i = 0; i < lista.size(); i++) {
    Elemento elemento = (Elemento)lista.get(i);
    anadirPalabra(elemento, palabras, texto);
  }
  listasPalabras.set(categoria, palabras);
}

/**
 * Método que añade una palabra a una lista de palabras
 * @param elemento Elemento Elemento que contiene la palabra que vamos a
 * @param palabras LinkedList Lista (de ParRaizLista) que contiene las
 * de palabras
 * @param texto int Número de texto al que corresponde la palabra
 */
private void anadirPalabra(Elemento elemento, LinkedList palabras,
                           int texto) {
  if(!anadirRaiz(elemento, palabras, texto)) {
    ParTextoFrecuencia parTF = new ParTextoFrecuencia(texto,
        elemento.getValor());
    LinkedList lista = new LinkedList();
    lista.addLast(parTF);
    ParRaizLista parRL = new ParRaizLista(elemento.getAtributo(), lista);
    palabras.addLast(parRL);
  }
}

/**
 * Método que añade una raíz a una lista de palabras
 * @param elemento Elemento Elemento que contiene la raiz que vamos a añadir
 * @param palabras LinkedList Lista (de ParRaizLista) que contiene las
 * de palabras
 * @param texto int Número de texto al que corresponde la raíz
 * @return boolean Devuelve true si la palabra ya estaba en la lista, false
 */
private boolean anadirRaiz(Elemento elemento, LinkedList palabras,
                           int texto) {
```

```
  boolean encontrado = false;
  int i = 0;
  String raiz = elemento.getAtributo();
  while(i < palabras.size() && !encontrado) {
    ParRaizLista par = (ParRaizLista)palabras.get(i);
    String raizPar = par.getRaiz();
    if(raiz.equals(raizPar)) {
      encontrado = true;
      ParTextoFrecuencia parTF = new ParTextoFrecuencia(texto,
          elemento.getValor());
      par.getLista().addLast(parTF);
    }
    i++;
  }
  return encontrado;
}

/**
 * Método que genera los pesos de las raices y los añade a listaPesos
 */
private void generarPesos() {
  try {
    cambiarOcurrenciasPesos();
    for(int i = 0; i < categorias.size(); i++) {
      LinkedList listaPesos = generarPesosCategoria((LinkedList)
          listasPalabras.
          get(i));
      listasPesos.addLast(listaPesos);
    }
  }
  catch(Exception e) {
    mostrarError("ERROR DESCONOCIDO", e.getMessage());
  }
}

/**
 * Método que cambia las frecuencias por los pesos calculados como
 * peso = frec * log(ntd / nda) * log(2);
 */
private void cambiarOcurrenciasPesos() {
  for(int i = 0; i < categorias.size(); i++) {
    LinkedList listaPalabras = (LinkedList)listasPalabras.get(i);
    int ntd = ((LinkedList)textos.get(i)).size();
    for(int j = 0; j < listaPalabras.size(); j++) {
      LinkedList frecuencias = ((ParRaizLista)listaPalabras.get(j)).
          getLista();
      int nda = frecuencias.size();
      for(int k = 0; k < frecuencias.size(); k++) {
        double frec = ((ParTextoFrecuencia)frecuencias.get(k)).
            getFrecuencia();
        double peso = frec * Math.log((ntd / nda) + 1) * Math.log(2);
        ((ParTextoFrecuencia)frecuencias.get(k)).setFrecuencia(peso);
      }
    }
  }
}

/**
 * Método que genera los pesos de una categoría
 * @param palabras LinkedList Lista (de ParRaizLista) que contiene los pesos
 * @return LinkedList Devuelve una lista (de ParRaizPeso) con los nuevos
 */
private LinkedList generarPesosCategoria(LinkedList palabras) {
  LinkedList listaPesos = new LinkedList();
  for(int i = 0; i < palabras.size(); i++) {
    ParRaizLista parRL = (ParRaizLista)palabras.get(i);
    LinkedList lista = parRL.getLista();
    double peso = 0;
```

```
    for(int j = 0; j < lista.size(); j++) {
      peso += ((ParTextoFrecuencia)lista.get(j)).getFrecuencia();
    }
    peso = peso / (lista.size());
    ParRaizPeso parRP = new ParRaizPeso(parRL.getRaiz(), peso);
    listaPesos.addLast(parRP);
  }
  return listaPesos;
}

/**
 * Método que guarda los resultados del entrenamiento en un archivo de
 * (".cat") y archivos de datos (".dat"), uno por categoría
 */
private void guardarResultados() {
  sfc.setToolTipText("Guarda el archivo de categorías");
  sfc.setDialogTitle("Guardar archivo");
  sfc.setDialogType(0);
  sfc.setCurrentDirectory(new File(ruta + "/training/"));
  sfc.setFileFilter(new FiltroCAT());
  int returnVal = sfc.showSaveDialog(null);
  String nombreFichero = "";
  if(returnVal == JFileChooser.APPROVE_OPTION) {
    nombreFichero = sfc.getSelectedFile().getAbsolutePath();
    String filtroAplicado = sfc.getFileFilter().getClass().getName();
    if(!nombreFichero.endsWith(".cat") &&
       filtroAplicado.compareTo("ri.FiltroCAT") == 0) {
      nombreFichero += ".cat";
    }
    guardarResultados(nombreFichero);
  }
}

/**
 * Metodo que guarda el resultado del entrenamiento
 * @param nombreFichero String Nombre del archivo de categorías (".cat") en
 * que se guardarán los resultados.
 */
private void guardarResultados(String nombreFichero) {
  File file = new File(nombreFichero);
  String directorio = file.getParent();
  String contenido = "########################################" +
      "\n# Archivo generado de forma automática #" +
      "\n########################################";
  contenido += "\n\n# Número de categorías\n";
  contenido += "NUM_CATEGORIAS = " + categorias.size() + "\n\n";
  for(int i = 0; i < categorias.size(); i++) {
    contenido += "\n####\n#### Datos de la categoría " + (i + 1) +
        "\n####\n";
    contenido += "NOMBRE_CAT_" + (i + 1) + " = " + categorias.get(i) +
        "\n";
    String archivo = obtenerDireccionRelativa((String)categorias.get(i));
    contenido += "DATOS_CAT_" + (i + 1) + " = " + archivo + "\n";
  }
  guardarFichero(nombreFichero, contenido);
  for(int i = 0; i < categorias.size(); i++) {
    String archivo = obtenerDireccion(directorio,
                                      (String)categorias.get(i));
    LinkedList lista = ((LinkedList)listasPesos.get(i));
    String contenidoDatos = "";
    for(int j = 0; j < lista.size(); j++) {
      ParRaizPeso parRP = (ParRaizPeso)lista.get(j);
      String raiz = parRP.getRaiz();
      double peso = parRP.getPeso();
      contenidoDatos += raiz + " " + peso + "\n";
    }
    guardarFichero(archivo, contenidoDatos);
  }
```

```
}

/**
 * Genera la dirección de los archivos de texto que contienen datos de cada
 * @param directorio String Directorio donde se guardarán estos archivos
 * @param nombreCategoria String Nombre de la categoría a la que pertenece
 * @return String Devuelve la ruta del archivo de texto.
 */
private String obtenerDireccion(String directorio,
                                String nombreCategoria) {
  String direccion = directorio;
  direccion += "\\";
  nombreCategoria = nombreCategoria.toLowerCase().replaceAll(" ", "_");
  direccion += nombreCategoria;
  direccion += ".dat";
  direccion = tratarDireccion(direccion);
  return direccion;
}

/**
 * Genera la dirección de los archivos de texto que contienen datos de cada
 * @param nombreCategoria String Nombre de la categoría a la que pertenece
 * @return String Devuelve la ruta del archivo de texto.
 */
private String obtenerDireccionRelativa(String nombreCategoria) {
  String direccion = "";
  direccion += "\\";
  nombreCategoria = nombreCategoria.toLowerCase().replaceAll(" ", "_");
  direccion += nombreCategoria;
  direccion += ".dat";
  direccion = tratarDireccion(direccion);
  return direccion;
}

/**
 * Trata la ruta de los archivos sustituyendo el caracter "\" por "\\" para
 * evitar problemas a la hora de ser leído.
 * @param direccion String Dirección que queremos tratar
 * @return String Devuelve la nueva dirección ya tratada
 */
private String tratarDireccion(String direccion) {
  String nuevaDireccion = "";
  for(int i = 0; i < direccion.length(); i++) {
    char c = direccion.charAt(i);
    nuevaDireccion += c;
    if(c == '\\') {
      nuevaDireccion += c;
    }
  }
  return nuevaDireccion;
}

/**
 * Guarda un String en un fichero de texto especificado
 * @param contenido String La cadena aque deseamos guardar en un fichero de
 * @param nombreFichero String Nombre del fichero en el que deseamos guardar
 * la cadena
 */
public void guardarFichero(String nombreFichero, String contenido) {
  FileWriter archivo;
  try {
    archivo = new FileWriter(nombreFichero);
    PrintWriter escritura = new PrintWriter(new BufferedWriter(archivo));
    escritura.println(contenido);
    escritura.close();
  }
  catch(IOException e1) {
```

```
      mostrarError("ERROR de escritura",
                   "Error al guardar en el archivo " + nombreFichero);
    }
    catch(Exception e2) {
      mostrarError("ERROR de escritura",
                   "Error al guardar en el archivo " + nombreFichero);
    }
  }

  /**
   * Muestra una ventana con un error.
   * @param titulo String Título de la ventana
   * @param mensaje String Mensaje de error que aparecerá en la ventana
   */
  private void mostrarError(String titulo, String mensaje) {
    JOptionPane.showMessageDialog(null, mensaje, titulo,
                                  JOptionPane.ERROR_MESSAGE);
  }

}
```

2.2. ri.Clasificador

```
package ri;

import java.util.*;

/**
 *
 * <p>Title: Recuperador de Información</p>
 *
 * <p>Description: Recuperador de Información</p>
 *
 * <p>Copyright: Copyright (c) 2005</p>
 *
 * <p>Company: </p>
 *
 * @author M. Fuertes, P. Gallo, N. Ortiz de Zárate
 * @version 1.0
 */
public class Clasificador {
  private LinkedList listaVectores; // Lista de vectores (uno por cada
  private LinkedList categorias = new LinkedList(); // Lista de categorías
  private LinkedList resultado = new LinkedList(); // Lista donde guardamos el

  /**
   * Constructora de la clase
   */
  public Clasificador() {
  }

  /**
   * Método que clasifica un texto en la categoría correspondiente
   * @param texto String El texto que queremos clasificar
   * @param datos LinkedList Lista (de LinkedList de ParRaizPeso) de vectores
   * las categorías (un vector por cada categoría)
   * @param cat LinkedList Lista (de String) de los nombres de las categorías
   * @return LinkedList Devuelve una lista (de ParCategoriaSimilitud) ordenada
   * por similitud con el texto de las categorías
   */
  public LinkedList clasificar(String texto, LinkedList datos,
                               LinkedList cat) {
    listaVectores = datos;
    categorias = cat;
    Transformador transformador = new Transformador();
    LinkedList listaRaices = transformador.analizar(texto);
    for(int i = 0; i < categorias.size(); i++) {
```

```
    double similitud = calculaSimilitud(listaRaices, i);
    ParCategoriaSimilitud parCS = new ParCategoriaSimilitud((String)
        categorias.get(i), similitud);
    anadeOrden(parCS);
  }
  return resultado;
}

/**
 * Método que calcula la similitud entre una categoría y un texto.
 * @param listaRaices LinkedList Lista (de Elemento) de las raices y sus
 * que representan el texto que queremos calcular su similitud
 * @param numCategoria int Número de categoría con la que queremos calcular
 * similitud
 * @return double Devuelve un valor entre 0 y 1 correspondiente con la
 */
private double calculaSimilitud(LinkedList listaRaices, int numCategoria) {
  LinkedList vector = (LinkedList)listaVectores.get(numCategoria);
  listaRaices = adaptarRaices(listaRaices, vector);
  double moduloA = calculaModulo(vector);
  double moduloB = calculaModulo(listaRaices);
  double producto = productoVectores(vector, listaRaices);
  return(producto / (moduloA * moduloB));
}

/**
 * Método que calcula el producto escalar entre dos vectores de ParRaizPeso
 * @param vector1 LinkedList Primer vector (lista de ParRaizPeso)
 * @param vector2 LinkedList Segundo vector (lista de ParRaizPeso)
 * @return double Devuelve el proucto escalar de ambos vectores
 */
private double productoVectores(LinkedList vector1, LinkedList vector2) {
  double producto = 0.0;
  for(int i = 0; i < vector1.size(); i++) {
    double peso1 = ((ParRaizPeso)vector1.get(i)).getPeso();
    double peso2 = ((ParRaizPeso)vector2.get(i)).getPeso();
    producto += peso1 * peso2;
  }
  return producto;
}

/**
 * Método que calcula el módulo de un vector de ParRaizPeso
 * @param vector LinkedList Vector (lista de ParRaizPeso)
 * @return double Devuelve el módulo del vector
 */
private double calculaModulo(LinkedList vector) {
  int tamano = vector.size();
  double cuadrados = 0;
  for(int i = 0; i < tamano; i++) {
    double peso = ((ParRaizPeso)vector.get(i)).getPeso();
    cuadrados += Math.pow(peso, 2);
  }
  return Math.sqrt(cuadrados);
}

/**
 * Método que adapta un vector (lista de ParRaizPeso) a los ejes de otro
 * @param listaRaices LinkedList Vector que queremos adaptar
 * @param vector LinkedList Vector de referencia del que se tomaran los ejes
 * @return LinkedList Devuelve el vector listaRaices adaptado (los ejes que
 * no aparecían en el vector de referencia se han eliminado y los que
 * aparecían enéste pero no en listaRaices han sido añadidos con el valor 0)
 */
private LinkedList adaptarRaices(LinkedList listaRaices,
                                 LinkedList vector) {
  LinkedList nuevaListaRaices = new LinkedList();
  for(int i = 0; i < vector.size(); i++) {
```

```
      String raiz = ((ParRaizPeso)vector.get(i)).getRaiz();
      double peso = devuelvePeso(listaRaices, raiz);
      ParRaizPeso parRP = new ParRaizPeso(raiz, peso);
      nuevaListaRaices.addLast(parRP);
    }
    return nuevaListaRaices;
  }

  /**
   * Método que devuelve el peso de una raiz en una lista de ParPesoRaiz
   * @param listaRaices LinkedList Lista de ParPesoRaiz en la que buscaremos
   * peso de la raíz
   * @param raiz String Raíz de la que queremos obtener su peso
   * @return double Devuelve el peso de la raíz
   */
  private double devuelvePeso(LinkedList listaRaices, String raiz) {
    boolean encontrado = false;
    int i = 0;
    double peso = 0.0;
    while(!encontrado && i < listaRaices.size()) {
      String raizActual = ((Elemento)listaRaices.get(i)).getAtributo();
      if(raiz.equals(raizActual)) {
        encontrado = true;
        peso = ((Elemento)listaRaices.get(i)).getValor();
      }
      else {
        i++;
      }
    }
    return peso;
  }

  /**
   * Método que añade un elemento en orden a la lista (de
   * ParCategoriaSimilitud)de resultados
   * @param par ParCategoriaSimilitud Elemento que queremos añadir a la lista
   */
  private void anadeOrden(ParCategoriaSimilitud par) {
    double similitudPar = par.getSimilitud();
    int i = 0;
    boolean encontrado = false;
    while(i < resultado.size() && !encontrado) {
      double similitudElemento = ((ParCategoriaSimilitud)resultado.get(i)).
          getSimilitud();
      if(similitudElemento < similitudPar) {
        encontrado = true;
      }
      else {
        i++;
      }
    }
    resultado.add(i, par);
  }
}
```

2.3. ri.Transformador

```
package ri;

import java.util.*;
import java.io.*;

/**
 * <p>Title: </p>
 *
 * <p>Description: Recuperador de Información</p>
 *
```

```
 * <p>Copyright: Copyright (c) 2005</p>
 *
 * <p>Company: </p>
 *
 * @author M. Fuertes, P. Gallo, N. Ortiz de Zárate
 * @version 1.0
 */
public class Transformador {
  private String texto = "";
  private LinkedList stopList = new LinkedList();
  private LinkedList listaRaices = new LinkedList();
  private static final String archivoPropiedades = "ri.properties";

  /**
   * Constructora de la clase sin parámetros
   */
  public Transformador() {
    try {
      Properties props = new Properties();
      props.load(new FileInputStream(archivoPropiedades));
      String ficheroStopList = props.getProperty("StopListPath");
      stopList = leerFichero(ficheroStopList);
    }
    catch(FileNotFoundException exception1) {
      System.out.println(
          "No se ha podido encontrar el archivo de propiedades " +
          archivoPropiedades);
    }
    catch(IOException exception2) {
      System.out.println("Error al cargar el archivo de propiedades " +
                         archivoPropiedades);
    }
  }

  /**
   * Constructora de la clase con un parámetro
   * @param t String El texto que se va a transformar
   */
  public Transformador(String t) {
    try {
      Properties props = new Properties();
      props.load(new FileInputStream(archivoPropiedades));
      String ficheroStopList = props.getProperty("StopListPath");
      stopList = leerFichero(ficheroStopList);
      texto = t;
    }
    catch(FileNotFoundException exception1) {
      System.out.println(
          "No se ha podido encontrar el archivo de propiedades " +
          archivoPropiedades);
    }
    catch(IOException exception2) {
      System.out.println("Error al cargar el archivo de propiedades " +
                         archivoPropiedades);
    }
  }

  /**
   * Método accesor para el atributo texto
   * @return String Devuelve el texto a transformar
   */
  public String getTexto() {
    return texto;
  }

  /**
   * Método modificador para el atributo texto
   * @param t String El nuevo texto a transformar
```

```
 */
public void setTexto(String t) {
  texto = t;
}

/**
 * Método accesor para el atributo listaRaices
 * @return LinkedList Devuelve la lista de las raices que aparecen en el
 * texto junto con el número de veces que aparecen
 */
public LinkedList getListaRaices() {
  return listaRaices;
}

/**
 * Método modificador para el atributo listaRaices
 * @param lista LinkedList La nueva lista de raices
 */
public void setListaRaices(LinkedList lista) {
  listaRaices = lista;
}

/**
 * Método que dado un texto devuelve una lista de ClaseAtributo con las
 * raíces de las palabras que aparecen en el texto y sus ocurrencias
 * @param entrada String Texto de entrada que se va a analizar
 * @return LinkedList Devuelve la lista de ClaseAtributo
 */
public LinkedList analizar(String entrada) {
  entrada = tratarEntrada(entrada);
  StringTokenizer st = new StringTokenizer(entrada);
  String token;
  LinkedList listaTokens = new LinkedList();
  while(st.hasMoreTokens()) {
    token = st.nextToken();
    if(!esPalabraVacia(token)) {
      token = obtenerRaiz(token);
      if(!token.equals("")) {
        listaTokens.add(token);
      }
    }
  }
  /** Devuelve la lista de tokens ordenados
   **/
  listaTokens = ordenarLista(listaTokens);
  listaTokens = transformarLista(listaTokens);
  listaRaices = listaTokens;
  return listaTokens;
}

/**
 * Método que dada una palabra devuelve su raiz
 * @param palabra String La palabra de la que queremos obtener su raiz
 * @return String La raiz de la palabra
 */
private String obtenerRaiz(String palabra) {
  String idioma = "eng";
  String raiz = palabra;
  try {
    Properties props = new Properties();
    props.load(new FileInputStream("ri.properties"));
    idioma = props.getProperty("language");
  }
  catch(IOException excep) {

  }
  if(idioma.equals("eng")) {
    Stemmer stemmer = new Stemmer(palabra);
```

```
    raiz = stemmer.stem();
  }
  else if(idioma.equals("esp")) {
    StemmerEsp stemmerEsp = new StemmerEsp();
    raiz = stemmerEsp.raiz(palabra);
  }
  return raiz;
}

/**
 * Método que trata el texto a analizar, dejándolo exclusivamente con los
 * caracteres que son letras o separadores. Además transforma todo el texto
 * dejándolo en minúsculas.
 * @param entrada String Texto que se va a tratar
 * @return String Devuelve el texto ya tratado con sólo palabras en minúscul
 */
private String tratarEntrada(String entrada) {
  /** El entero i lleva el número de caracteres ya tratados
   **/
  int i = 0;
  /** Se trata la cadena de entrada hasta el final, es decir, cuando el
   * contador i llega al número de caracteres de la misma
   **/
  while(i < entrada.length()) {
    /** El caracter c almacena el caracter actual
     **/
    char c = entrada.charAt(i);
    String principio;
    String fin;
    if(!Character.isLetter(c) && !Character.isSpaceChar(c)) {
      principio = entrada.substring(0, i);
      principio += " ";
      fin = entrada.substring(i + 1);
      entrada = principio.concat(fin);
      i--;
    }
    i++;
  }
  entrada = entrada.toLowerCase();
  return entrada;
}

/**
 * Método que nos dice si una palabra está vacía de significado comprobando
 * si está o no está en la lista de palabras vacías
 * @param palabra String Palabra que se va a comprobar
 * @return boolean Devuelve true si es una palabra vacía y false si no lo es
 */
private boolean esPalabraVacia(String palabra) {
  palabra = palabra.toLowerCase();
  return stopList.contains(palabra);
}

/**
 * Método que dada una lista la ordena según el orden de sus elementos
 * (orden establecido por la función "compareTo()" de cada elemento)
 * @param lista LinkedList Lista que va a ser ordenada
 * @return LinkedList Devuelve la lista ya ordenada
 */
private LinkedList ordenarLista(LinkedList lista) {
  Object[] tabla;
  tabla = (Object[])lista.toArray();
  Arrays.sort(tabla);
  LinkedList listaOrdenada = new LinkedList();
  for(int i = 0; i < tabla.length; i++) {
    listaOrdenada.add(tabla[i]);
  }
  return listaOrdenada;
```

```
}

/**
 * Método que transforma la lista de raíces en una lista de ClaseAtributo
 * con la raíz y el tanto por mil de las ocurrencias
 * @param lista LinkedList Lista de entrada que va a ser transformada
 * @return LinkedList Devuelve la lista ya transformada
 */
private LinkedList transformarLista(LinkedList lista) {
  LinkedList listaTransformada = new LinkedList();
  int total = lista.size();
  while(!lista.isEmpty()) {
    String primero = lista.removeFirst().toString();
    double ocurrencias = 1;
    String siguiente = "";
    if(lista.size() >= 2) {
      siguiente = lista.getFirst().toString();
    }
    while((primero.compareTo(siguiente) == 0) && (!lista.isEmpty())) {
      ocurrencias++;
      lista.removeFirst();
      if(!lista.isEmpty()) {
        siguiente = lista.getFirst().toString();
      }
    }
    Elemento elemento = new Elemento(primero, ocurrencias / total);
    if(primero.compareTo("") != 0) {
      listaTransformada.add(elemento);
    }
  }
  return listaTransformada;
}

/**
 * Método que lee el fichero de palabras vacías y crea la lista de palabras
 * sin significado
 * @param nombreArchivo String Nombre del archivo que contiene la lista de
 * las palabras carentes de carga semántica
 * @return LinkedList Devuelve la lista de las palabras vacías
 */
private static LinkedList leerFichero(String nombreArchivo) {
  /** Captura excepciones por si el archivo no puede ser abierto normalmente
   **/
  try {
    FileInputStream archivo = new FileInputStream(nombreArchivo);
    int tamanno = archivo.available();
    byte buff[] = new byte[tamanno];
    int i = archivo.read(buff);
    String cadena = new String(buff);
    StringTokenizer st = new StringTokenizer(cadena);
    String palabra;
    LinkedList listaPalabras = new LinkedList();
    while(st.hasMoreTokens()) {
      palabra = st.nextToken();
      listaPalabras.add(palabra);
    }
    return listaPalabras;
  }
  /** Muestra un mensaje de error en el caso de que no se haya podido abrir
   **/
  catch(IOException e1) {
    System.out.println("Error al abrir el archivo " + nombreArchivo);
    return null;
  }
  catch(Exception e2) {
    System.out.println("ERROR");
    return null;
  }
```

```
  }
}
```

3. ASAUD

3.1. GeneradorDescriptores

```
package ri;

import java.io.*;
import java.util.*;

import javax.swing.*;
import java.sql.*;

/**
 * <p>Title: Recuperador de Información</p>
 *
 * <p>Description: Recuperador de Información</p>
 *
 * <p>Copyright: Copyright (c) 2005</p>
 *
 * <p>Company: </p>
 *
 * @author M. Fuertes, P. Gallo, N. Ortiz de Zárate
 * @version 1.0
 */
public class GeneradorDescriptores {

  private LinkedList categorias;
  private LinkedList datos;
  private static final String archivoPropiedades = "generador.properties";
  private static String archivoCat = "";
  private static String archivoSalida = "";
  private static String nombreTabla = "";
  private static int porcentajeCorte = 50;
  private static int minDescriptores = 1;
  private static int maxDescriptores = 3;
  private static int tiempoEspera = 1000;
  private static String bdDriver = "";
  private static String bdPath = "";
  private static String bdName = "";
  private static String bdPass = "";
  private static String salida = "";
  private ConexionBD conexion;

  /**
   * Constructora de la clase
   */
  public GeneradorDescriptores() {
    try {
      Properties props = new Properties();
      props.load(new FileInputStream(archivoPropiedades));
      archivoCat = props.getProperty("categorias");
      archivoSalida = props.getProperty("archivoSalida");
      nombreTabla = props.getProperty("nombreTabla");
      porcentajeCorte = Integer.parseInt(props.getProperty(
          "porcentajeCorte"));
      minDescriptores = Integer.parseInt(props.getProperty(
          "minDescriptores"));
      maxDescriptores = Integer.parseInt(props.getProperty(
          "maxDescriptores"));
      tiempoEspera = Integer.parseInt(props.getProperty("tiempoEspera"));
      bdDriver = props.getProperty("bdDriver");
      bdPath = props.getProperty("bdPath");
      bdName = props.getProperty("bdName");
      bdPass = props.getProperty("bdPass");
      categorias = new LinkedList();
      datos = new LinkedList();
```

```
      mostrarConfiguracion();
    }
    catch(IOException excep) {
      mostrarError("ERROR: I/O Exception", excep.getMessage());
    }
  }

  /**
   * Método que muestra por pantalla la configuración de la aplicación
   */
  private void mostrarConfiguracion() {
    System.out.println("ASIGNADOR AUTOMATICO DE DESCRIPTORES (AsAuD)");
    System.out.println("");
    System.out.println("Datos de la configuracion: ");
    System.out.println("Archivo .cat: " + archivoCat);
    System.out.println(
        "Porcentaje de similitud que deben tener los descriptores: " +
        porcentajeCorte + "%");
    System.out.println("Numero minimo de descriptores: " + minDescriptores);
    System.out.println("Numero maximo de descriptores: " + maxDescriptores);
    System.out.println("Base de datos: ");
    System.out.println("\tDriver: " + bdDriver);
    System.out.println("\tPath: " + bdPath);
    System.out.println("\tName: " + bdName);
    System.out.println("\tPassword: " + bdPass);
    System.out.println("");
  }

  /**
   * Método que genera los descriptores de las asignaturas y guarda en un
archivo
   * las consultas SQL necesarias para introducirlos en la base de datos.
   */
  public void generarDescriptores() {
    System.out.println("Asignando descriptores...");
    System.out.print("\tCargando archivo de categorias...");
    cargarCategorias(archivoCat);
    System.out.println(" hecho");
    System.out.print("\tObteniendo los codigos de las asignaturas...");
    LinkedList listaCodigos = obtenerListaCodigos();
    System.out.println(" hecho");
    System.out.println("\tObteniendo los programas de las asignaturas...");
    LinkedList listaProgramas = obtenerListaProgramas(listaCodigos);
    System.out.println("\tAsignando descriptores a las asignaturas...");
    int numAsignaturas = listaCodigos.size();
    for(int i = 0; i < numAsignaturas; i++) { //Se hace para cada asignatura
      String programa = (String)listaProgramas.get(i);
      String codigoAsignatura = (String)listaCodigos.get(i);
      System.out.print("\t\tAsignando descriptores a " + codigoAsignatura +
                       "...");
      LinkedList resultado = obtenerDescriptores(programa);
      String consultas = generarConsultas(codigoAsignatura, resultado);
      salida += consultas;
      salida += "\n";
    }
    System.out.print("Guardando consultas en " + archivoSalida + " ...");
    guardarFichero(archivoSalida, salida);
    System.out.println(" hecho");
    System.out.println(
        "\nLa asignacion de descriptores ha finalizado con exito");
    try {
      System.in.read();
    }
    catch(IOException e) {}
  }

  /**
   * Método que obtiene la lista de los códigos de asignaturas a las que
```

```
 * queremos asignar descriptores
 * @return Devuelve la lista de los códigos de las asignaturas
 */
private LinkedList obtenerListaCodigos() {
  LinkedList lista = new LinkedList();
  lista = getCodigos();
  return lista;
}

/**
 * Método que devuleve la lista de los programas de las asignaturas cuyo
 * código se encuentra en la lista de códigos listaCodigos
 * @param listaCodigos Lista de los códigos de las asignaturas
 * @return Devuelve la lista de los programas de las asignaturas
 */
private LinkedList obtenerListaProgramas(LinkedList listaCodigos) {
  LinkedList listaProgramas = new LinkedList();
  JTextPane pane = new JTextPane();
  JTextArea area = new JTextArea();
  for(int i = 0; i < listaCodigos.size(); i++) {
    String cod = (String)listaCodigos.get(i);
    String URL = obtenerURL(cod);
    System.out.print("\t\tObteniendo el programa de la asignatura " +
                     cod + "...");
    if(URL != null) {
      try {
        pane = new JTextPane();
        pane.setPage(URL);
        Thread.sleep(tiempoEspera);
        area.setDocument(pane.getDocument());
        String programa = area.getText();
        listaProgramas.addLast(programa);
        System.out.println(" hecho");
      }
      catch(IOException excep1) {
        System.out.println(" ERROR: No se puede cargar la página: " +
                           URL);
      }
      catch(Exception excep2) {
        System.out.println(" **ERROR**");
      }
    }
    else {
      System.out.println(" ERROR: No se puede cargar el programa de " +
                         cod + " (URL es null)");
    }
  }
  return listaProgramas;
}

/**
 * Método que dado un código de asignatura devuelve la URL del programa
 * @param codigo Codigo de la asignatura
 * @return Devuelve un String con la URL del programa de la asignatura.
 */
private String obtenerURL(String codigo) {
  String url = getURL(codigo);
  return url;
}

/**
 * Genera las consultas SQL, para insertar los descriptores en la base de
 * datos
 * @param codigo Codigo de la asignatura
 * @param resultado LinkedList Lista de descriptores
 * @return String Devuelve las consultas SQL para insertar los descriptores
 * en la base de datos
 */
```

```
private String generarConsultas(String codigo, LinkedList resultado) {
  String consulta = "";
  for(int i = 0; i < resultado.size(); i++) {
    String descriptor = ((ParCategoriaSimilitud)resultado.get(i)).
        getCategoria();
    consulta += "INSERT INTO " + nombreTabla + " VALUES (\"" + codigo +
        "\", \"" + descriptor + "\"); -- AUTOMATIC VALUE (NOT CHECKED)";
    consulta += "\n";
  }
  return consulta;
}

/**
 * Dado un texto devuleve la lista de descriptores que se le asignan
 *
 * @param texto String Texto al que se le quieren asignar descriptores
 * @return LinkedList
 */
private LinkedList obtenerDescriptores(String texto) {
  LinkedList resultado = new LinkedList();
  if(!texto.equals("") && categorias.size() > 0) {
    Clasificador clasificador = new Clasificador();
    resultado = clasificador.clasificar(texto, datos, categorias);
    resultado = filtrarResultado(resultado);
    System.out.println(" hecho");
  }
  else if(categorias.size() <= 0) {
    System.out.println(" ERROR: No se han cargado las categorías");
    /*mostrarError("ERROR al clasificar",
                 "Primero hay que cargar las categorías");*/
  }
  else {
    System.out.println(" ERROR: Programa vacío");
    /*mostrarError("ERROR al clasificar",
                 "No se puede clasificar un texto vacío");*/
  }
  return resultado;
}

/**
 * Método que filtra la lista de categorias-similitud dejando sólo las que
 * pasan un umbral (porcentajeCorte). En cualquier caso la lista tendrá un
 * tamaño entre minDescriptores y maxDescriptores.
 *
 * @param lista LinkedList Lista que vamos a filtrar
 * @return LinkedList
 */
private LinkedList filtrarResultado(LinkedList lista) {
  LinkedList mejores = new LinkedList();
  if(minDescriptores < lista.size()) {
    int limite = Math.min(maxDescriptores, lista.size());
    int i = 0;
    while(i < limite) {
      double similitud = ((ParCategoriaSimilitud)lista.get(i)).
          getSimilitud();
      if(similitud < (porcentajeCorte / 100.0)) {
        i = limite;
      }
      else {
        mejores.addLast((ParCategoriaSimilitud)lista.get(i));
      }
      i++;
    }
  }
  return mejores;
}

/**
```

```
 * Carga las categorías de un archivo de categorías (".cat")
 * @param nombreArchivo String La ruta del archivo de categorías
 */
private void cargarCategorias(String nombreArchivo) {
  try {
    Properties props = new Properties();
    props.load(new FileInputStream(nombreArchivo));
    int numCategorias = Integer.parseInt(props.getProperty(
        "NUM_CATEGORIAS"));
    categorias.clear();
    datos.clear();
    for(int i = 0; i < numCategorias; i++) {
      String nombreCat = props.getProperty("NOMBRE_CAT_" + (i + 1));
      categorias.addLast(nombreCat);
      String archivoDatos = props.getProperty("DATOS_CAT_" + (i + 1));
      File fich = new File(nombreArchivo);
      String direc = fich.getParent();
      String ruta = direc + archivoDatos;
      String datosString = abrirFichero(ruta);
      LinkedList datosLista = cargarDatosCategoria(datosString,
          archivoDatos);
      datos.addLast(datosLista);
    }
  }
  catch(FileNotFoundException excep1) {
    mostrarError("ERROR al cargar archivo",
                 "No se ha podido encontrar el archivo " + nombreArchivo);
  }
  catch(IOException excep2) {
    mostrarError("ERROR al cargar archivo",
                 "Se ha producido un error al abrir el archivo " +
                 nombreArchivo);
  }
}

/**
 * Carga los datos de un archivo de datos (".dat") en un vector de ParRaizPe
 * @param datos String El contenido del archivo de datos
 * @param nombreArchivo String La ruta del archivo de datos
 * @return LinkedList Devuelve el vector de ParRaizPeso guardado en el archi
 */
private LinkedList cargarDatosCategoria(String datos,
                                        String nombreArchivo) {
  LinkedList lista = new LinkedList();
  try {
    StringTokenizer st = new StringTokenizer(datos);
    while(st.hasMoreElements()) {
      String raiz = st.nextToken();
      double peso = Float.parseFloat(st.nextToken());
      ParRaizPeso parRP = new ParRaizPeso(raiz, peso);
      lista.addLast(parRP);
    }
  }
  catch(Exception excep) {
    mostrarError("ERROR al cargar datos",
                 "Error al cargar los datos desde " + nombreArchivo);
  }
  return lista;
}

/**
 * Guarda un String en un fichero de texto especificado
 * @param contenido String La cadena aque deseamos guardar en un fichero de
 * @param nombreFichero String Nombre del fichero en el que deseamos guardar
 * la cadena
 */
public void guardarFichero(String nombreFichero, String contenido) {
  FileWriter archivo;
```

```
    try {
      archivo = new FileWriter(nombreFichero);
      PrintWriter escritura = new PrintWriter(new BufferedWriter(archivo));
      escritura.println(contenido);
      escritura.close();
    }
    catch(IOException e1) {
      mostrarError("ERROR de escritura",
                   "Error al guardar en el archivo " + nombreFichero);
    }
    catch(Exception e2) {
      mostrarError("ERROR de escritura",
                   "Error al guardar en el archivo " + nombreFichero);
    }
  }

  /**
   * Abre un fichero de texto devolviendo su contenido
   * @param nombreArchivo String Nombre del fichero que deseamos abrir
   * @return String Devuelve el contenido del fichero que deseamos abrir
   * @throws IOException Lanza una excepción si se produce algún error al
   * abrir el archivo
   */
  public String abrirFichero(String nombreArchivo) throws IOException {
    try {
      FileInputStream archivo = new FileInputStream(nombreArchivo);
      int tamanno = archivo.available();
      byte buff[] = new byte[tamanno];
      int i = archivo.read(buff);
      String cadena = new String(buff);
      return cadena;
    }
    catch(IOException e1) {
      mostrarError("ERROR",
                   "Error al abrir el archivo de entrada " + nombreArchivo);
      throw new IOException();
    }
    catch(Exception e2) {
      mostrarError("ERROR",
                   "Error al abrir el archivo de entrada " + nombreArchivo);
      return null;
    }
  }

  /**
   * Muestra una ventana con un error.
   * @param titulo String Título de la ventana
   * @param mensaje String Mensaje de error que aparecerá en la ventana
   */
  private void mostrarError(String titulo, String mensaje) {
    JOptionPane.showMessageDialog(null, mensaje, titulo,
                                  JOptionPane.ERROR_MESSAGE);
  }

  /**
   * Método que devuelve los códigos de las asignaturas de la base de datos
   * @return LinkedList Devuelve una lista con los códigos de las asignaturas
   */
  public LinkedList getCodigos() {
    conexion = new ConexionBD(bdDriver, bdPath, bdName, bdPass);
    LinkedList lista = new LinkedList();
    try {
      Statement stmt = conexion.creaSentencia();
      ResultSet rset = stmt.executeQuery("SELECT Code FROM Subject;");
      //Vamos almacenando en la lista los resultados obtenidos
      while(rset.next()) {
        String codigo = (String)rset.getObject("Code");
        lista.addLast(codigo);
```

```
      }
    }
    catch(SQLException e) {
      System.out.println(e);
      conexion.cerrarConexion();
    }
    conexion.cerrarConexion();
    return lista;
  }

  /**
   * Método que accede a la base de datos para obtener la dirección de un prog
   * @param codigo String Código de la asignatura de la que vamos a obtener
   * el programa
   * @return String Devuelve un String con la URL del programa
   */
  public String getURL(String codigo) {
    conexion = new ConexionBD(bdDriver, bdPath, bdName, bdPass);
    String url = "";
    //SELECT Programa FROM Subject WHERE (Code = 'Codigo');
    try {
      Statement stmt = conexion.creaSentencia();
      ResultSet rset = stmt.executeQuery(
          "SELECT Programa FROM Subject WHERE (Code = '" + codigo + "');");
      //Vamos almacenando en la lista los resultados obtenidos
      while(rset.next()) {
        url = (String)rset.getObject("Programa");
        break;
      }
    }
    catch(SQLException e) {
      System.out.println(e);
      conexion.cerrarConexion();
    }
    return url;
  }

  /**
   * Metodo principal de la clase
   * @param args String[] No tiene argumentos
   */
  public static void main(String[] args) {
    GeneradorDescriptores generador = new GeneradorDescriptores();
    generador.generarDescriptores();
  }
}
```